AD A043015

RADC-TR-77-369, Volume II (of three)
Final Technical Report
November 1977

FACTORS IN SOFTWARE QUALITY
Metric Data Collection and Validation

Jim A. McCall
Paul K. Richards
Gene F. Walters

General Electric Company

AD No.
DDC FILE COPY

Approved for public release; distribution unlimited.

DDC
RECEIVED
JAN 23 1978
D

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report contains a large percentage of machine-produced copy which is not of the highest printing quality but because of economical consideration, it was determined in the best interest of the government that they be used in this publication.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-77-369, Vol II (of three) has been reviewed and approved for publication.

APPROVED: Joseph P. Cavano

JOSEPH P. CAVANO
Project Engineer

APPROVED: Alan R Barnum

ALAN R. BARNUM, Assistant Chief
Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIS) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A049055

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-77-369	2. GOVT ACCESSION NO.: Vol-2
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): FACTORS IN SOFTWARE QUALITY, Volume II. Metric Data Collection and Validation,	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. Aug 76 - Jul 77,
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Jim A. McCall, Paul K. Richards, Gene F. Walters	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0417
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Electric ~~Command & Information Systems~~, 450 Persian Drive, Sunnyvale CA 94086	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 64740F, 22370301, 03
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441	12. REPORT DATE: November 1977; 13. NUMBER OF PAGES: 145
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (12 255p.)	15. SECURITY CLASS. (of this report): UNCLASSIFIED; 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer:
Joseph P. Cavano (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Quality
Quality Factors
Metrics
Software Measurements

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An hierarchical definition of factors affecting software quality was compiled after an extensive literature search. The definition covers the complete range of software development and is broken down into non-oriented and software-oriented characteristics. For the lowest level of the software-oriented factors, metrics were developed that would be independent of the programming language. These measurable criteria were collected and validated using actual Air Force data bases. A handbook was generated that will be useful to Air Force

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

149450

AB

acquisition managers for specifying the overall quality of a software system.

PREFACE

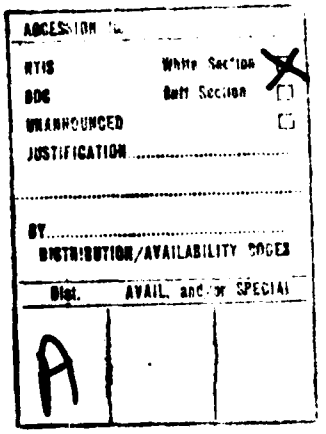
This document is the final technical report (CDRL A003) for the Factors in Software Quality Study, contract number F030602-76-C-0417. The contract was performed in support of the U.S. Air Force Electronic Systems Division's (ESD) and Rome Air Development Center's (RADC) mission to provide standards and technical guidance to software acquisition managers.

The report consists of three volumes, as follows:

Volume I	Concept and Definitions of Software Quality
Volume II	Metric Data Collection and Validation
Volume III	Preliminary Handbook on Software Quality for an Acquisition Manager

The objective of the study was to establish a concept of software quality and provide an Air Force acquisition manager with a mechanism to quantitatively specify and measure the desired level of quality in a software product. Software metrics provide the mechanism for the quantitative specification and measurement of quality.

This second volume describes the application of the metrics to software products and the validation of the metrics' relationship to software quality.



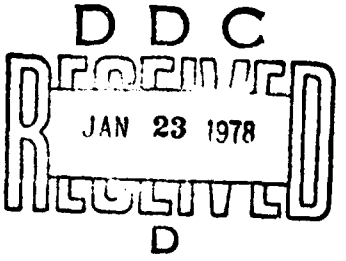


TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

SECTION 7

RELATIONSHIP OF METRICS TO QUALITY FACTORS

7.1 CONCEPT OF RELATIONSHIP

The hierarchical framework which has evolved supports the simple, understandable, and logical relationships of the components of the software quality concept. It also supports the mathematical formulation of the relationship between the metrics and the quality factors.

A software development can be envisioned as a process which is controlled by management (both contractor management and AF SPO personnel). This control is exercised through reviews, status reporting, and software products (Section 5, Interim Report #2) delivered during the development effort (Figure 7.1-1). Currently, the major emphasis of the control is to evaluate the schedule and cost performance and to determine the functional correctness of the software being developed. The concept underlying software quality metrics is to use these control vehicles to provide an indication (and therefore a mechanism of control) of the quality of the software product to be delivered.

These software qualities or characteristics which go beyond the technical mission - qualities such as reliability, maintainability, usability, testability, and portability, which have been defined in previous sections - have been recognized in recent years as a necessary concern for software development managers.

This recognition has come about because of many instances in which not considering factors such as these has driven total project costs well beyond initial estimates. It has been found that the costs throughout the total life cycle are more affected by the characteristics of the software system than by the mission-oriented functions performed by the software system. Large software systems have sometimes proven untestable, unmodifiable, and largely unusable by operations personnel because of the characteristics of the software.

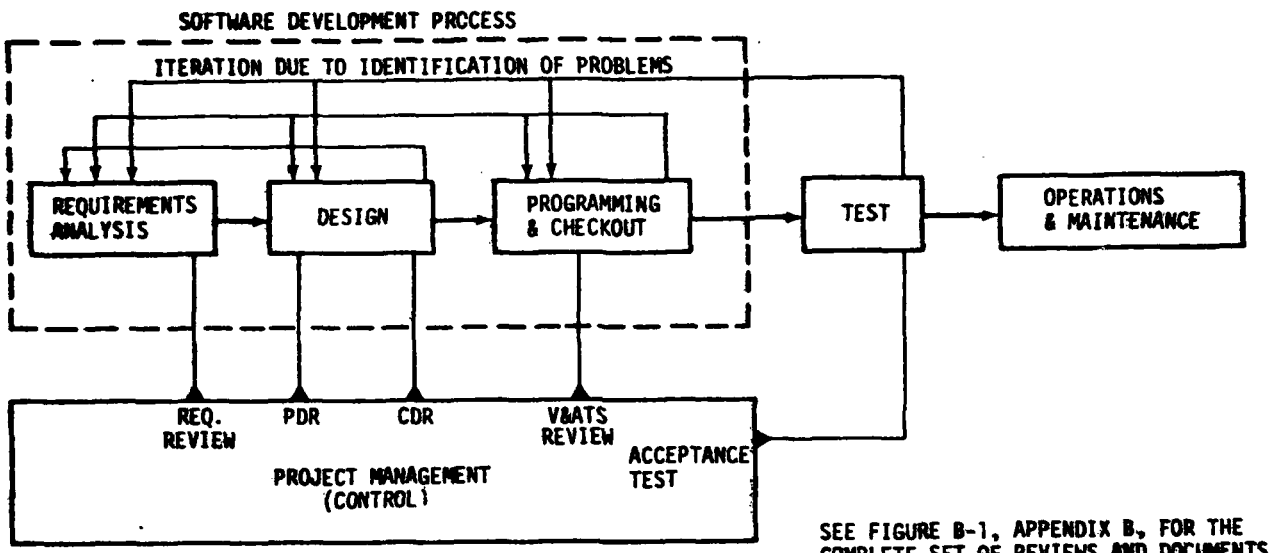


Figure 7.1-1 Software Development Process Control

The metrics that have been established provide quantitative measures of specific software attributes. At any specific time during the software development, a set of metrics can be applied to available review material, documents, and code. Table 6.2-1 identifies which metrics were applied during the three phases of requirements analysis, design, and programming.

When the metrics are applied, the resulting measurements can be viewed as an n-tuple:

$$(m_1, m_2, m_3, \ldots, m_n)$$

Each element, m_i, of this n-tuple represents a quantitative measure of the system with respect to a specific metric or software attribute.

Certain subsets of this n-tuple relate to specific software quality factors. For example, metrics i = 1 to k may relate to maintainability. The subset of the n-tuple

$$(m_1, m_2, \ldots, m_k)$$

then, are the metrics which Table 6.3-1 identifies as related to maintainability. This relationship can be viewed as a function relating the measurement-tuple to a rating of the specific quality factor:

$$f(m_1, m_2, \ldots, m_k) = r_M$$

where r_M is a rating of the maintainability of the software.

The definitions of the quality factors support the concept of a rating, e.g., the rating for the quality factor, maintainability, would be in terms of the amount of effort (man-days) required to maintain the software.

A preliminary identification of the nature of the relationship f( ) was the goal of the fourth phase of this contract effort. The relationship is called the normalization function.

The derivation of mathematically complete, generally applicable functions were beyond the scope of this effort. The limiting factor was the size and nature of our sample. This aspect of the study is discussed in paragraph 7.2. The procedure

for the derivation of the normalization functions (paragraph 7.3), the concept of their use through a figure of merit procedure (paragraph 7.5), and the methodology of validating these relationships (paragraph 7.4) are discussed in this report and specific examples from our experience are presented.

The metric n-tuple has considerable value from a quality assurance viewpoint. An analogy can be drawn with the set of indicator lights in a cockpit of an airplane. If a particular indicator light flashes, this immediately identifies a specific characteristic which is beyond acceptable limits or has reached a level at which attention should be focused on that characteristic. There may be a sound reason for the indicator to be flashing, not necessarily resulting from an underlying problem, but a justification should be established. This particular use of the metric n-tuple, without the precise relationship provided by a normalization function, is explored and discussed further in the conclusion of this report and in Volume III.

7.2 DATA USED FOR THIS STUDY

Two large-scale software system developments for the Air Force were used as the data bases for application of the metrics. The two systems were chosen because they represent large-scale software developments, were developed according to military standards, and represent two different applications. Figure 7.2-1 presents some of the general characteristics of the two systems.

System A is a command and control system developed in JOVIAL (J4). Periodically, a new version of the system is developed in response to changes in hardware and software requirements. The data base used represents a recent formal product delivery of the system. The system represents an application with which both the customer and GE have had considerable experience. The system has an excellent operational history.

System B is a data base management system developed in JOVIAL (J4). The data base used is the initial software development of the system. It typified the development of a new capability and exhibited several significant problems during the design and development phases. The system provides the capabilities

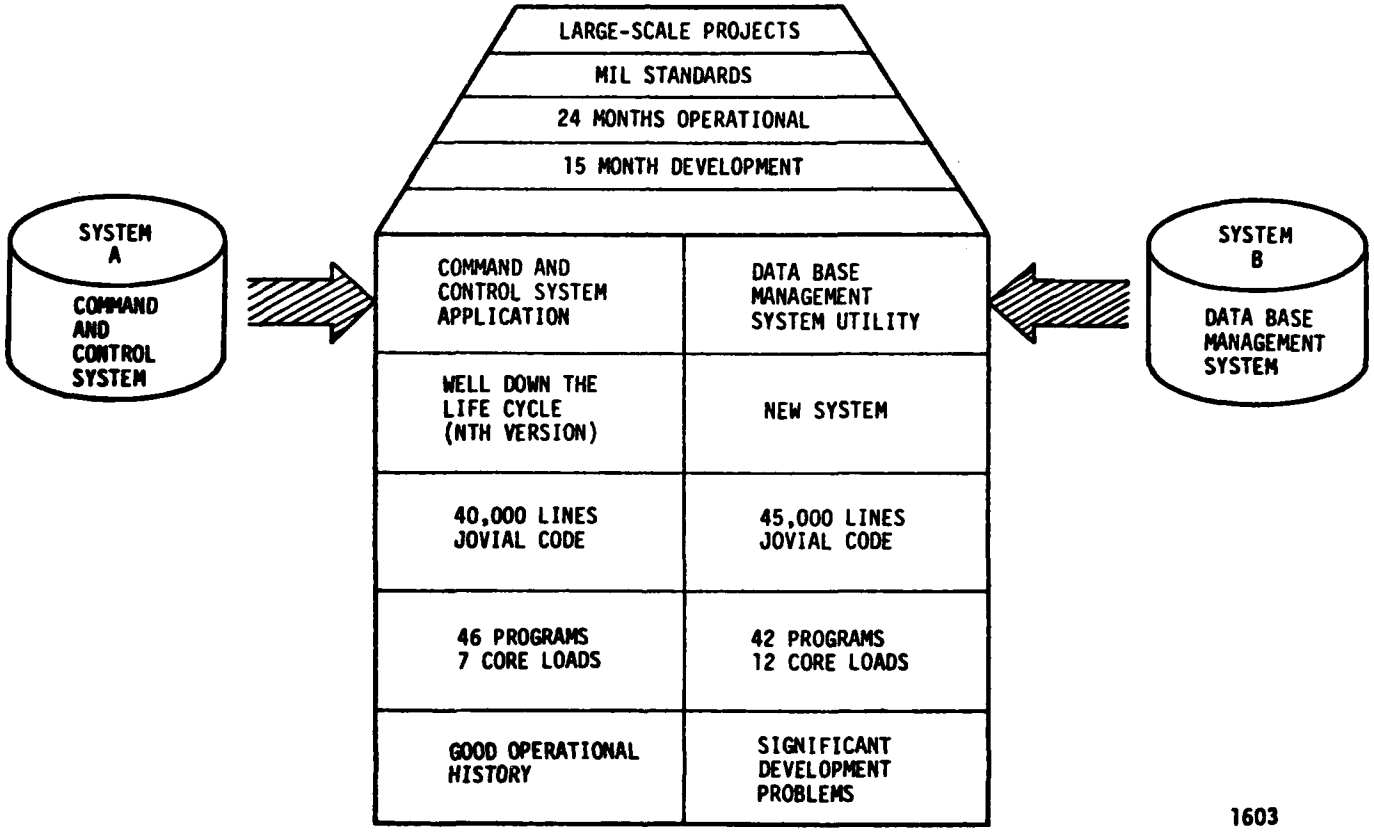


Figure 7.2-1 Air Force's Data Bases Chosen

to update, delete, and modify portions of a large data base as well as selectively list, retrieve, or compare two data bases.

The data base for each system consists of the following:

- Documents - The complete set of documents identified in Appendix B, including the requirements specification, design specifications, test plans, user manual, interface control document, etc. were available for analysis.
- Review Material - The documentation prepared for and the recorded proceedings of the various reviews identifed in Appendix B were part of the data base.
- Source Code - The source code listings of each program in both systems were available both in hardcopy and on magnetic tape.
- Problem Reports and Configuration Management Reports - The complete set of problem reports identified in Appendix B were available as well as the configuration management report which maintains a log and status of the problem reports. Figure 7.2-2 presents examples of each of these reports.

  The Design Problem Report, DPR (item G), identifies problems the programmer and customer have encountered with the system analyst's design. The Software Problem Report, SPR (item C), identifies software problems encountered with a program. The Software Analysis Report, SAR (item D), provides the scope of the problem and recommends the solution/action. The Modification Transmittal Memorandum, MTM (item E), COMPOOL Change Request, CCR (item F), and the Data Base Change Request, (DBCR, not shown), indicate solutions/changes implemented. The Configuration Management (CM) Status Update Listing (item A) provides time-to-fix statistics by tracking the maintenance efforts. The Summary Listing (item B) will be discussed in more detail in Section 8 since it provided automated metric data.

BEST AVAILABLE COPY

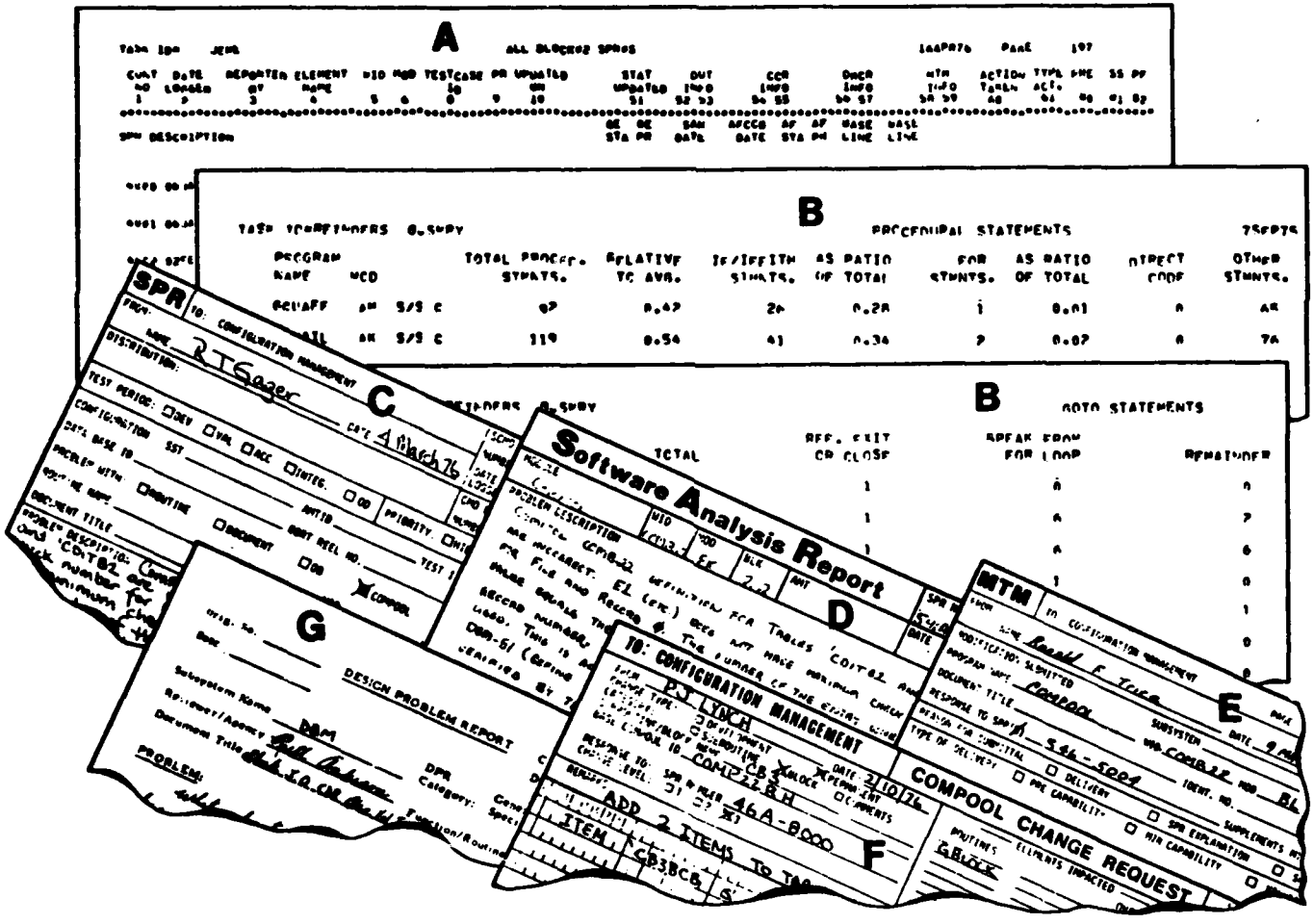


Figure 7.2-2 Problem Reports and Configuration Management

The problem reports were used to establish ratings for the various quality factors. For the two data bases, a significant number of reports were available, as illustrated in Figure 7.2-3.

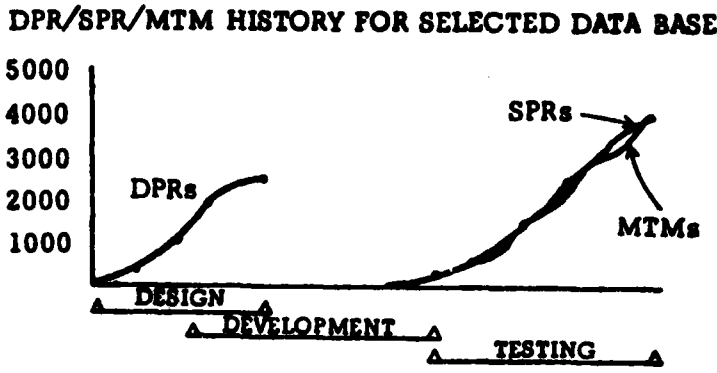


Figure 7.2-3 DPR/SPR/MTM History For Selected Data Bases

For the selected sample of modules, the corresponding problem reports were extracted and categorized. The categorization was accomplished by analyzing the problem and solution described on the problem reports and grouping the problem according to Table 7.2-1. Also, data on man-power expenditure to fix problems, to make changes, etc. was extracted from the configuration management system and categorized according to Table 7.2-1. It was felt that man-power expenditures represented the severity or cost of problems more than the number of problems.

- Design Charts - As part of the design documents, overview and detailed design charts are provided. For System B, these design charts were available in machine readable form, which facilitated automated metric analysis.
- Metric Information - Considerable metric data was available as part of the data bases. The summary listings (item B in Figure 7.2-2) provided a statistical profile of each program for the two systems. Such statistics as the number of cards, statements, procedures, declaratives, comments, IFs, FORs, direct code statements (assembly

Table 7.2-1 Problem Report and Man-Power Expenditure Categorization

CATEGORY BY QUALITY FACTOR	EXPLANATION
• CORRECTNESS	The function which the software is to perform is incorrect. The rating is in terms of effort required to fix.
• RELIABILITY	The software does not function as expected. The rating is in terms of effort required to fix.
• EFFICIENCY	The software does not meet performance (speed, storage) requirements. The rating is in terms of effort required to fix.
• INTEGRITY	The software does not provide required security. The rating is in terms of effort required to fix.
• USABILITY	There is a problem related to operation of the software, the user interface, or the input/output. The rating is in terms of effort required to fix.
• MAINTAINABILITY	The rating is in terms of effort required to correct any of the above problems.
• FLEXIBILITY	The rating is in terms of effort required to make a modification due to a change in specifications.
• TESTABILITY	The rating is in terms of effort required to test changes or fixes.
• REUSABILITY	The rating is in terms of effort required to use software in a different application.
• PORTABILITY	The rating is in terms of effort required to convert the software to operate in a different environment.
• INTEROPERABILITY	The rating is in terms of effort required to couple the system to another system.

language), GOTOs, breaks from loops, operands, operators, delimiters, etc. were available for the two systems. This metric data was oriented primarily to the source code. This metric data, as well as all of the other metrics established during this effort, are covered in more detail in Section 8.

Each of these items was available for analysis. Essentially, they were utilized as sources for metric data or, in the case of problem reports and the configuration management report, as sources of the error and maintenance history of the software. While our data bases represent an extremely comprehensive set of data about a software system development, some difficulties were encountered. Several previous or on-going efforts ([THAYT76], [NELSR75], [SHOOM75], [WILLN76]) sponsored by RADC have very ably discussed the problems of data collection. Basically, the following problems arise because the data is collected after the fact:

1. Large volume of data which must be manually analyzed
2. Completeness and validity of data with respect to goal of analysis is difficult to determine.
3. Impact on production process and personnel must be kept at a minimum.
4. Interpretation of data decreases in accuracy as the age of the data increases.

The impact of item 1, with the resources available in this study, was to reduce the number of modules that were analyzed. System-level metrics were applied to both systems but module-level metrics were only applied to approximately 40% of the modules of both systems. The subset of modules were chosen to be representative of the systems. An equal distribution of modules which were small (< 400 statements), medium ($400 \leq n \leq 800$ statements), and large (≥ 800 statements) in size and which had a small number of SPRs (< 10), a medium number ($10 \leq n \leq 25$), and a large number (> 25) written against them were chosen.

The impact of items 2, 3, and 4 above on our study was minimal because of the large amount of data and complete documentation that is collected and generated during a software development in our environment.

Several other restrictions were imposed on the study because of data availability. During the derivation of the quality factors and quality metrics, a complete view of software was taken. However, the operational and maintenance historical data necessary to validate all of the quality factors was not available. For example, some of the metrics are system-level metrics only, i.e., they are measured at the system level. Since only two system developments were used, development of a normalization function and validation of its accuracy was not possible. Also, the two systems have not experienced all of the activities required to accumulate historical data to validate metrics relating to several of the quality factors. For example, neither system has been converted to operate in another environment. Therefore, a normalization function relating the metrics with the quality factor portability was not possible. Metrics relating to interoperability, portability, reusability, testability, integrity, and efficiency could not be analyzed because historical data was not available.

All of the metrics were applied to obtain experience with their data collection. This experience is described in Section 8 and Appendix D. Table 7.2-1 identifies which quality factors and their related metrics were supported by the data available. The plus (+) sign indicates that an analysis was possible because data was available and the metric could be applied at the module level. A zero (0) indicates that either the metric was a system level metric and therefore the sample was too small and/or historical data was not available to conduct an analysis. The code column relates the metrics to the software quality metrics table 6.2-1.

Table 7.2-2 Data Support for Normalization Function Development and Validation

METRICS	CODE	LEVEL AT WHICH METRIC APPLIED	QUALITY FACTORS										
			CORRECTNESS	RELIABILITY	EFFICIENCY	USABILITY	INTEGRITY	MAINTAINABILITY	FLEXIBILITY	TESTABILITY	REUSABILITY	PORTABILITY	INTEROPERABILITY
			A	A	NA	PA	NA	A	PA	NA	NA	NA	NA
TRACEABILITY	TR.1	SYS	0										
COMPLETENESS CHECKLIST	CP.1	SYS	0										
PROCEDURE CONSISTENCY	CS.1	SYS	0	0				0					
DATA CONSISTENCY	CS.2	SYS	0	0				0					
ACCURACY CHECKLIST	AY.1	SYS		0									
ERROR TOLERANCE CONTROL	ET.1	SYS		0									
ERROR TOLERANCE INPUT DATA	ET.2	SYS		0									
ERROR TOLERANCE RECOVERY FROM COMPUTATIONAL FAILURES	ET.3	SYS		0									
ERROR TOLERANCE RECOVERY FROM HARDWARE FAULTS	ET.4	SYS		0									
ERROR TOLERANCE RECOVERY FROM DEVICE ERRORS	ET.5	SYS		0									
DESIGN STRUCTURE	SI.1	MOD		+				+		0			
STRUCTURE PROGRAMMING	SI.2	MOD		+				+		0			
COMPLEXITY MEASURE	SI.3	MOD		+				+		0			
CODE SIMPLICITY	SI.4	MOD		+				+		0			
STABILITY MEASURE	MO.1	SYS						0	0	0	0	0	0
MODULAR IMPLEMENTATION	MO.2	MOD						+	+	0	0	0	0
REFERENCE GENERALITY	GE.1	SYS							0		0		
IMPLEMENTATION GENERALITY	GE.2	MOD							+		0		
DATA STORAGE EXPANDABILITY	EX.1	SYS							0				
COMPUTATION EXTENSIBILITY	EX.2	SYS							0				
MODULE TESTING	IN.1	MOD								0			
INTEGRATION TESTING	IN.2	SYS								0			
SYSTEM TESTING	IN.3	SYS								0			
QUANTITY OF COMMENTS	SD.1	MOD						+	+	0	0	0	0
EFFECTIVENESS OF COMMENTS	SD.2	MOD						+	+	0	0	0	0
DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE	SD.3	MOD						+	+	0	0	0	0
PERFORMANCE REQUIREMENTS ALLOCATED TO DESIGN	EE.1	SYS			0								
ITERATIVE PROCESSING EFFICIENCY	EE.2	MOD			0								
DATA USAGE EFFICIENCY	EE.3	MOD			0								
STORAGE EFFICIENCY	SE.1	MOD			0								
ACCESS CONTROL	AC.1	SYS					0						
ACCESS AUDIT	AA.1	SYS					0						
OPERABILITY	OP.1	SYS				0							
TRAINING	TG.1	SYS				0							

Table 7.2-2 Data Support for Normalization Function Development and Validation (Cont.)

METRICS	CODE	LEVEL AT WHICH METRIC APPLIED	QUALITY FACTORS										
			CORRECTNESS	RELIABILITY	EFFICIENCY	USABILITY	INTEGRITY	MAINTAINABILITY	FLEXIBILITY	TESTABILITY	REUSABILITY	PORTABILITY	INTEROPERABILITY
			A	A	NA	PA	NA	A	PA	NA	NA	NA	NA
USER INPUT INTERFACE	CM.1	SYS				0							
USER OUTPUT INTERFACE	CM.2	SYS				0							
SOFTWARE SYSTEM INDEPENDENCE	SS.1	MOD									0	0	
MACHINE INDEPENDENCE	MI.1	MOD									0	0	
COMMUNICATIONS COMMONALITY	CC.1	SYS											0
DATA COMMONALITY	DC.1	SYS											0
CONCISENESS	CO.1	SYS						+					

LEGEND

SYS - SYSTEM-LEVEL METRIC
MOD - MODULE-LEVEL METRIC
A - DATA AVAILABLE
PA - DATA PARTIALLY AVAILABLE
NA - DATA NOT AVAILABLE

(A, PA, NA: HISTORICAL DATA AVAILABLE TO DEVELOP NORMALIZATION FUNCTION FOR QUALITY FACTOR)

\+ - ANALYSIS POSSIBLE
0 - ANALYSIS NOT POSSIBLE

A more subtle yet very significant impact on our study was the fact that we applied the metrics well after the system had been delivered and was operational. Many of the problems (low metric scores) which would have been realized had the metrics actually been applied during the development were not evident. For example, had we applied the set of metrics related to design at the time of the CDR, significantly different metric scores than the ones recorded in this study would have been realized. Over time, many of the problems have been identified, analyzed and corrected. This bias that was introduced is very significant, especially to the metrics applied during the requirements and design phases. The metric scores can be assumed to be significantly inflated.

Thus, the most effective and accurate means of applying the metrics and also establishing normalization functions would be in an on-line mode, that is, applying the metrics during a software development effort, tracking error history, and then accumulating operational and maintenance historical data to establish normalization functions. The data would be current and therefore more accurate and easier to collect and would reflect the status of the software development in terms of the software quality metrics more realistically. This on-line application of the metrics is described further in Section 8 when the application of the metrics during this effort is discussed.

7.3 NORMALIZATION FUNCTION DEVELOPMENT

In this section, a description of the methodology of deriving a normalization function will be described and then examples will be provided. Complete results from this effort are contained in Appendix C.

The methodology is as follows:

- The metric n-tuple for a particular phase of development is applied to the available software products (review material, documents, code). This process is done initially at a module-by-module basis and then at a system level. The application of each metric is described in Section 8. The results of this step are *n*-tuples of measurements for each module and for each system.

- Subsets of the measurement n-tuples which relate to specific quality factors are segregated. For example, the k-tuple, $(m_1, m_2, \ldots m_k)$, which represents the measurements for the k metrics which relate to the quality factor, maintainability, are extracted for each module and each system.
- Data which represent the quality factor performance or rating of the individual modules and systems are collected. For example, the amount of effort expended to correct fixes to each module and system is collected to represent a rating of the maintainability of that module or system.
- Using the measurement k-tuple as independent variables and the ratings of the individual modules or systems as dependent variables, a regression analysis is performed to derive the normalization function. Linear regression analysis was performed in this study. There is some indication that in a few selected cases a nonlinear function might be more appropriate. This exploration should be considered in future efforts. The resulting function, in the linear regression case, takes the form:

$$r_f^p = a_0 + a_1 m_1 + a_2 m_2 + \ldots a_k m_k$$

where r_f^p is the predicted rating of quality factor f, given the measurement k-tuple $(m_1, m_2, m_3, \ldots, m_k)$ and the a_i are the regression coefficients derived from the regression analysis. These weights assigned the individual metrics reflect their predictive value with respect to the particular quality factor. Several iterations of this procedure are required to eliminate the metrics which do not show significant correlation. If time had permitted, initially a complete factors analysis would have been performed to group related metrics with specific quality factors. This was done intuitively during the process of establishing the metrics.

There is a serious misinterpretation which can be made at this point. The utility of the derived normalization function is very dependent upon the sample used. In the case of this study, the two systems used,

while two different applications, were developed in the same environment, according to very strict standards and conventions, using the same language, machine, operating system, development tools, etc. For this reason many metrics, when applied to all of the modules, showed no variation in measurements. A simple example is the metric (SI.2), use of a structured language or structured preprocessor. JOVIAL (J4) was considered a structured language, although according to a very strict definition it is not, because;

- Several of the structured programming constructs are implemented within the language.
- It is a block oriented language
- Our standards and conventions restrict the use of constructs which violate some of the structured programming philosophies.

Every module, then, received a score, or measurement, of 1 for this metric. Since this measurement showed no variation, the regression analysis indicated there was no correlation between this metric and the quality factor rating.

If this result is interpreted absolutely, then one could conclude that the use of a structured language or structured preprocessor has no effect or correlation with the resulting maintainability of the system. This is obviously an incorrect conclusion. What the result does mean is that for the application in which the metrics were applied and the regression analysis was performed, the variability in the maintainability of the modules in a system or between systems is not a function of the use of a structured language. The reason is that the use of a structured language is a standard to which there is strict adherence.

Our expectation is that if these metrics and methodology were applied to other system developments in other environments and, for this particular example, in environments where the use of a structured language or structured preprocessor varied, a significant correlation between this metric and the resulting rating of maintainability would be indicated.

Thus, the use and interpretation of the normalization function is critical to its effectiveness. This concept is considered in more depth in Section 9.

To illustrate the methodology an example will be presented in this section. Complete data from the regression analyses are provided in Appendix C.

Using the quality factors which were supported with historical data and for the metrics which could be applied at the module level (see Table 7.2-2), several analyses were performed. These analyses included analyzing individual metrics versus the rating of a factor and a multiple regression analysis of the k-tuple of metrics relating to a factor. In certain cases individual elements of a metric were also analyzed on a single basis with a quality factor when high correlation was expected. The methodology to perform any one of these analyses was the same. An example of the analysis of one metric, SD.2, effectiveness of comments measure, and its relationship to the quality factor, maintainability, will be given to illustrate the methodology.

A sample of modules from System B was used to develop the normalization function relating metric, SD.2, individually to maintainability.

Standard linear regression techniques ([THAYT76], [FLEIT66], [LABOV66], [COOLW62] [POOLL77], [PADED56], [KUESJ73]) were used. Routines to perform the analysis we developed on a PDP 11/40 and Tektronix 4051 terminal. Since the metric and the quality factor rating were normalized positive values, all data points fall within the positive quadrant of a graph. The regression line was forced through the origin to support this concept.

Using the metric table, Table 6.2-1, the measures associated with the effectiveness of comments metric were collected from a sample of modules. Historical operations and development data was used to determine the number of fixes to each module in the sample and the number of man-days expended to accomplish these fixes. A rating of maintainability was then calculated for each module by the following formula:

$$r_M^i = 1/(\frac{MD_i}{n_i})$$

where:

MD_i = Total number of man-days expended on fixes to module i.

n_i = Total number of fixes to module i.

r_M^i = Normalized rating of maintainability for module i.

The rating of maintainability then is based on the average number of man-days expended to make a fix to the software.

For the sample chosen, the distribution of occurrence for the metric value and the rating are shown in Figure 7.3-1 and Figure 7.3-2 respectively. The histogram is generated by accumulating the number of data point falling in the interval $k < x_i \leq k + 0.1$, where $k = 0, 0.1, 0.2, \ldots, 0.9$ and dividing by the total number of modules in the sample to arrive at a frequency of occurrence figure.

The independent variable is the metric value determined for each module. The dependent variable is the rating value determined for each module. The resulting regression equation is the normalization function. Its form in the case of one independent variable is:

$$r_M = a_M m_i^e$$

where r_M is the predicted rating of maintainability, a_M is the predictor coefficient for the metric value, m_i^e, in this case metric SD.2, $m_i^{SD.2}$.

Figure 7.3-3 illustrates the regression line determined for this metric:

$$r_M = 0.46 m_i^{SD.2}$$

The dashed lines represent a 90 percent confidence interval for the sample. The standard error of estimate is 0.15. The correlation coefficient for the regression line is 0.92. This represents a significant correlation between the metric and the rating of maintainability.

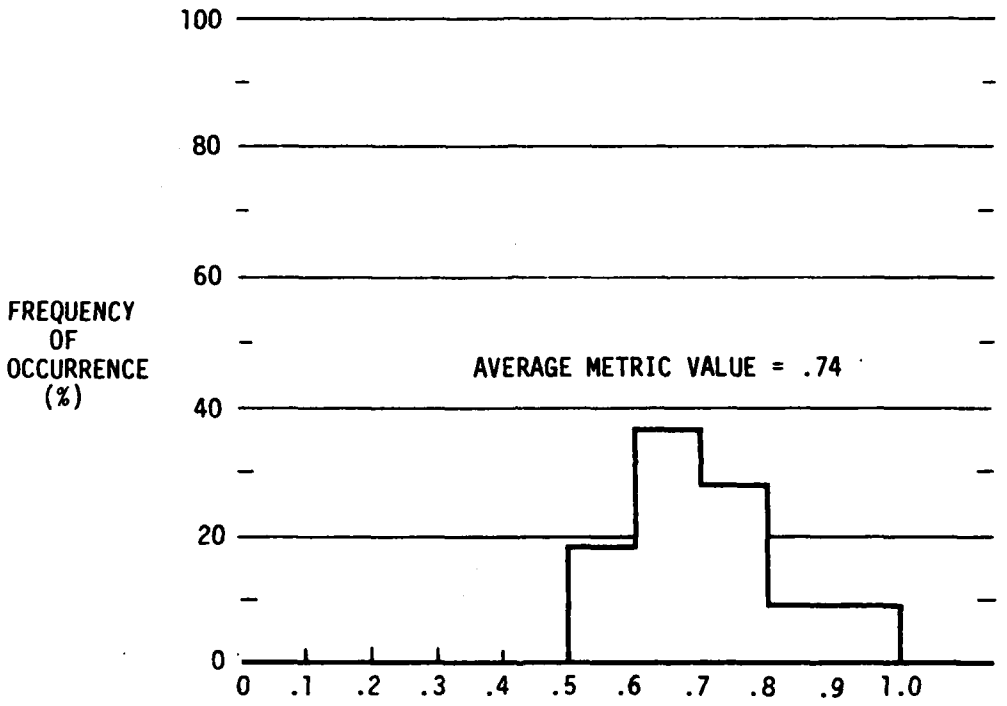
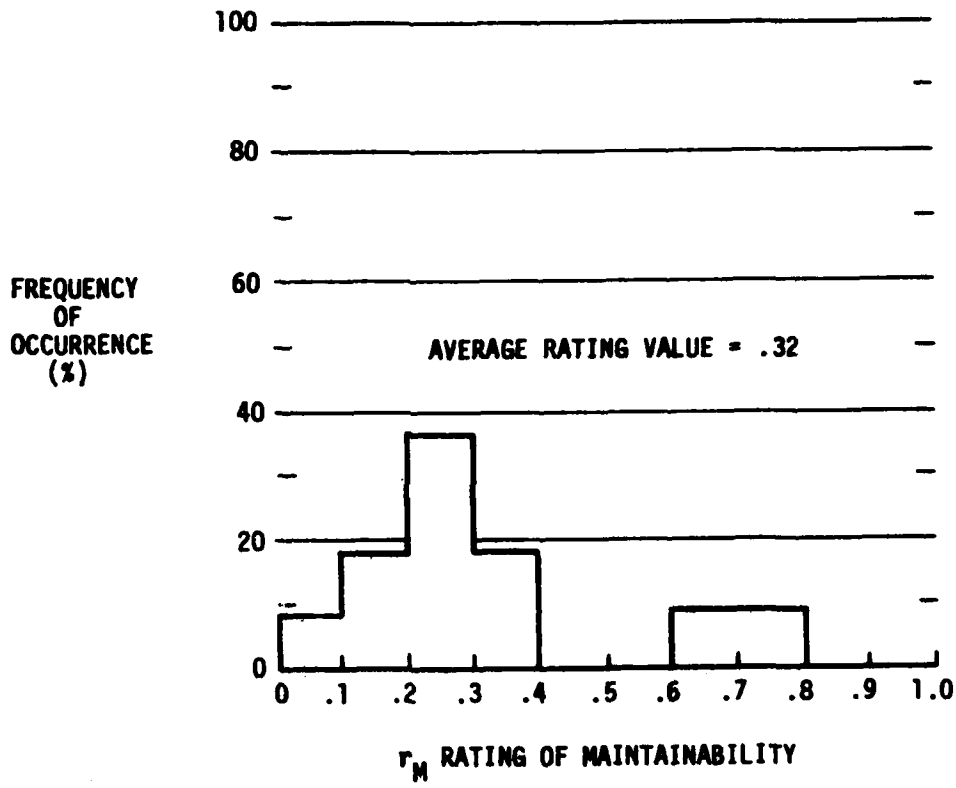
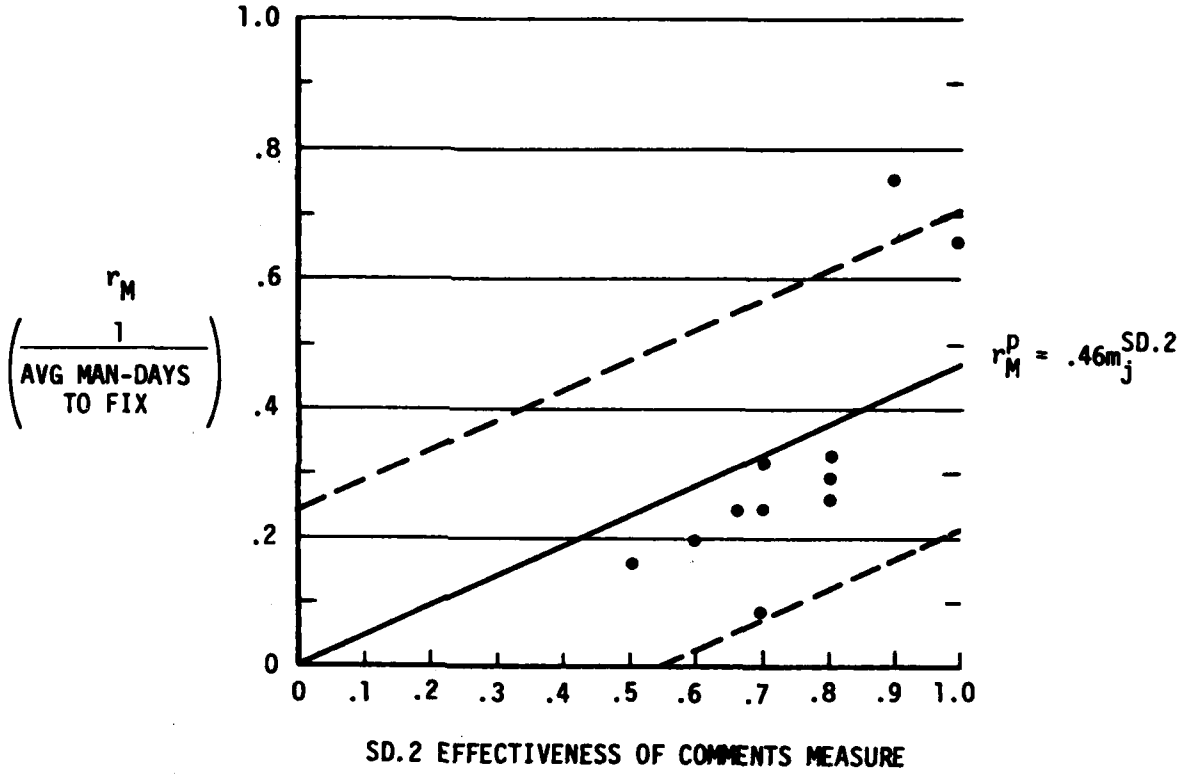


Figure 7.3-1 Frequency Distribution for Metric SD.2



1587

Figure 7.3-2 Frequency Distribution for r_M



1586

Figure 7.3-3 SD.2 (Implementation) Normalization Function

This information is displayed in Table 7.3-1. This table is an excerpt from Table C-8, where the results of the analyses of all of the metrics related to maintainability are described.

This example will be continued in paragraphs 7.4 and 7.5 to illustrate the methodology of validating the metrics and arriving at a figure of merit. The analysis of data points which fall outside the 90 percent confidence interval will be discussed in paragraph 7.4. The histograms are provided to allow some evaluation of the sample. In this example, all metric values were above 0.5. While we would have liked to had a better range of metric values for the sample, other modules measured in System B fall within this same range. This represents an environmental limitation to our study because comments are strongly emphasized by our standards and conventions.

Appendix C contains the remainder of the results of the normalization function development. Tables such as Table 7.3-1 provide summary results of the normalization functions for the groups of metrics related to a quality factor, for each individual metric related to a quality factor, and in some instances, selected metric elements related to a quality factor. Remarks highlight specific results in Appendix C. Where regression analysis was not performed, the summarized metric values are provided for information and there is an indication as to the reason why no regression analysis was performed.

Table 7.3-1 Regression Analysis Summary

SYSTEM B		METRIC SD.2	MAINTAINABILITY RATING
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.75	.32
	RANGE	.5-1.	.07-.77
	STD DEV	.13	.15
	PREDICTOR COEFFICIENT	.46	
	STANDARD ERROR OF ESTIMATE	.15	
	CORRELATION COEFFICIENT	.92	

7.4 VALIDATION PROCESS

The process of validating a normalization function will be described and examples presented. Appendix C contains a summary of the complete set of data used during the validation phase of this effort.

The methodology used for validating a normalization function is as follows:

- The normalization function derived in paragraph 7.3 predicts a rating for any measurement k-tuple. There is a certain variance associated with the predicted rating and actual rating.
- a 90% confidence interval is determined based on the normalization function, the variance, and the sample. Another subset of modules is then plotted and depending on their compliance with the confidence interval, the normalization function is accepted or rejected.
- In the case where a normalization function is rejected, several actions can be taken:
  - Conduct a factor analysis to determine the minimum number of dimensions needed to describe the relevant information contained in the original measurements.
  - Reevaluate the metrics and their units to ensure they do not present possible ambiguities in their measurement of the criterion or quality factor.
  - Evaluate the correlation coefficients of the metrics. While a metric may logically be an important characteristic of a software product, it may not correlate well and therefore is not a consistent predictor of the final quality of the software.
  - Reestablish a normalization function by utilizing regression analysis techniques.
  - Evaluate the quality factor rating (its representative distribution). It may not be linear itself and therefore cause problems in establishing a linear relationship with the associated metrics. Consideration of nonlinear regression will be given in this case.

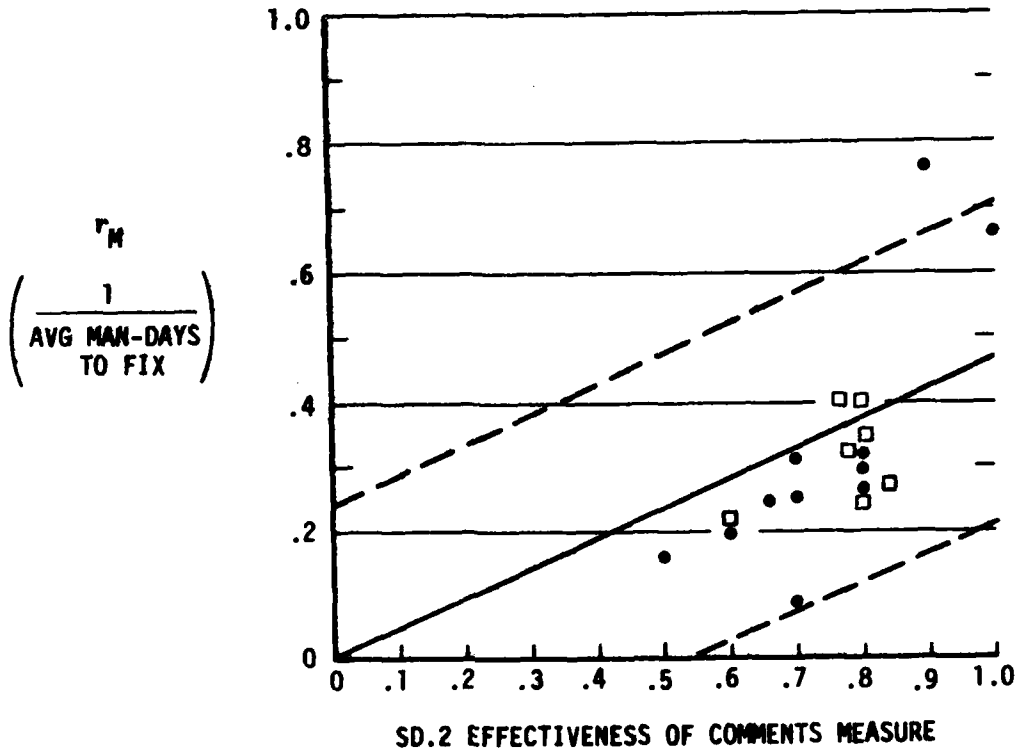
In the example presented in paragraph 7.3, a linear function for the metric SD.2, effectiveness of comments, and a 90% confidence interval were determined using a subset of modules of System B. The corresponding metric data and maintainability rating data were compiled for a subset of modules of System A. This data is summarized in Table 7.4-1.

Table 7.4-1 Validation Data Summary for Maintainability

SYSTEM A		METRIC SD.2
Individual metric analysis	Average	.77
	Range	.6-.8
	Std dev	.018
Rating	Average	.306
	Range	.21-.40
	Std dev	.068
	Normalization function accepted/ rejected	Accepted

Plotting the points on the graph presented in Figure 7.3-3 results in Figure 7.4-1. All of the data points fall within the 90% confidence interval. Recalculation of the regression line using all of the data points does not change any of the values substantially. Thus, this normalization function is considered validated.

Note that there was one point in the initial sample which fell outside the 90% confidence interval. An evaluation of this module revealed that while several fixes had been made to the module, they all were related to a basic problem. Thus, the average time to make the several fixes was quite low, resulting in the unusually high rating. Based on this explanation, we felt it was not justified to reject the predictor coefficient determined. During the validation for several metrics, data points fell outside of the 90% confidence interval. Those modules were evaluated for any abnormalities that would justify their deviation from the norm. Where justifications were not found, the normalization function was rejected. In the cases where time, resources, and data permitted, the steps described in the beginning of this paragraph were taken.



PREDICTOR COEFF:	.46
AVG m:	.74
RANGE OF m:	.5-1.0
STD DEV:	.13
AVG r:	.32
STD ERROR OF EST:	.15
CORR COEFF:	.92

1588

Figure 7.4-1 Validation of SD.2

The statements made in paragraph 7.2 qualifying the interpretation of the normalization function should be reemphasized here. The sample, both in size and variation in nature, limits the general applicability and precision of the results. The methodology established and the results achieved, nevertheless, are valuable.

For instance, of the three metrics quantifying self-descriptiveness (SD.1, SD.2, SD.3), only SD.2, effectiveness of comments, correlated well with maintainability. SD.1, a measure of the quantity of comments, varied insignificantly in our sample. The mean was .25 (the percent of comments per card was 25%), ranging essentially between .20 and .30. This lack of variation in the measurements meant no correlation with maintainability could be established. The reason for this lack of variation is our standards and conventions establish very strict guidelines on what situations should be commented. This does not mean that SD.1 would not be a valuable metric in another environment where the standards are not as strict. It does mean that in our environment, where the percentage of comments/cards in programs are fairly standard, no variation in maintainability is attributable to that metric.

SD.3, descriptiveness of implementation language, did not correlate well with maintainability either. Again, the main reason was the lack of variation in the measurements caused by our strict standards on the format of the programs and naming conventions for variables and a specific software support tool (GEDIT) used to preprocess all source code and indent, block, and number the code in a logical, standard manner. Variation in this metric would be found between system developments or in an environment where strict standards or automated tools are not used.

The summarized data and results of the validation process for these three metrics, as well as all of the other metrics for which normalization functions were derived, are in Appendix C.

7.5 FIGURE OF MERIT PROCEDURE

The intent in deriving a figure of merit is to provide the SPO with an overall measure for each quality factor. The figure of merit will be normalized according to the standard units of measurement chosen for that factor. In keeping with our example, the figure of merit for maintainability will be in terms of the average time to fix.

The figure of merit procedure is basically the use of the normalization function as a predictor of the level of quality being achieved for a quality factor. Thus, if a particular measurement k-tuple is applied at a specific time during the development phase, the values obtained can be inserted in the normalization function for that quality factor for that phase and a figure of merit is determined.

This figure of merit can then be evaluated relative to the level of quality specified by the customer. If the figure of merit is below the specified level, evaluation and/or corrective actions should be initiated. If the figure of merit is above the specified quality level, then some degree of confidence that the development effort is progressing satisfactorily with respect to the required software qualities is derived.

Figure 7.5-1 continues the example used previously. The normalization function is:

$$r_M = .46\ m_i^{SD.2}$$

If the system (or module) measurement for SD.2 was found to be .75 during the implementation phase of the development, a predicted rating of .345 results. If the SPO had specified that the required maintainability of the system was an average time-to-fix of one man-week (1/5 man-days = r_M = .2, identified in Figure 7.5-1 by point A), he has approximately an 84% level of confidence that the maintainability of his system will be better than he specified. This figure is arrived at using the predicted rating, the specified rating, and the standard error of estimate determined during the regression analysis. Figure 7.5-2 illustrates the derivation of the confidence level.

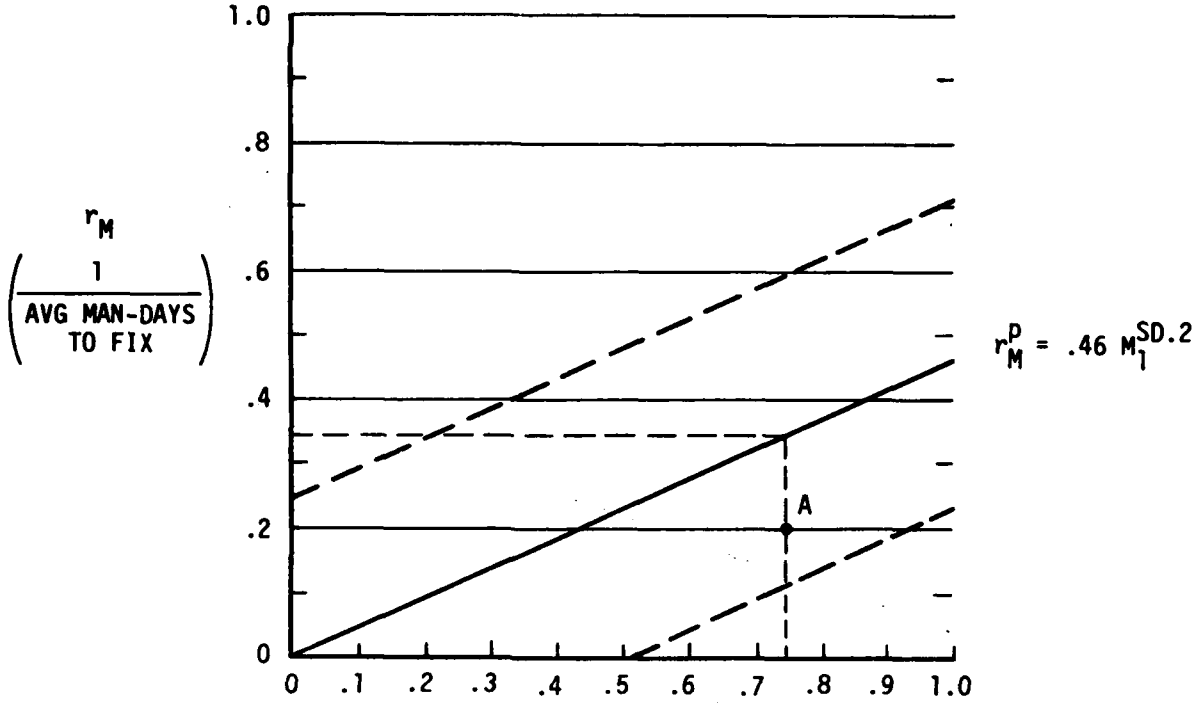
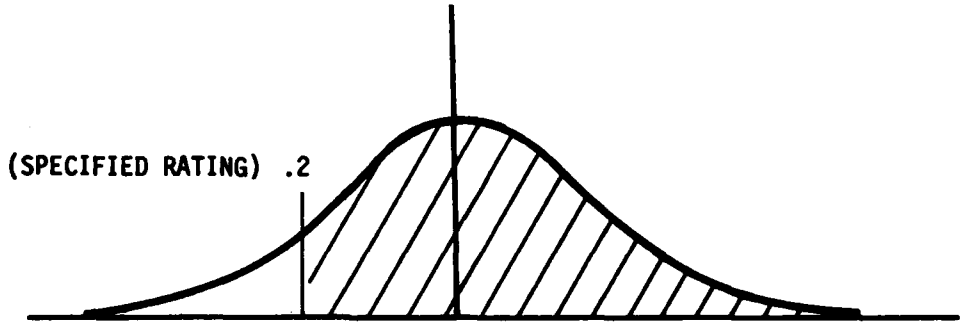


Figure 7.5-1 Figure of Merit Procedure



MEAN = .345 (PREDICTED RATING)

STANDARD DEVIATION = .15 (STANDARD ERROR OF ESTIMATE)

LEVEL OF CONFIDENCE = Pr $\{x \geq .2\}$ = .84 (SHADED AREA)

1590

Figure 7.5-2 Determination of Level of Confidence

This relationship between the figure of merit derived from the metrics and the quality level specified by an SPO is elaborated upon in Appendix E, which presents a preliminary handbook for SPOs on the specification of software qualities.

SECTION 8

METRIC DATA COLLECTION

8.1 METRIC APPLICATION

While the historical data and sample size prevented the establishment of normalization functions for all of the quality factors, a significant benefit of the study was the experience gained in applying all of the metrics to the systems. Whether regression analysis was to be performed or not, the metric data was collected for all of the metrics.

It was found that applying the metrics to the various software products provided considerable insight into the design and implementation of the software. This reflects the n-tuple (indicator lights) concept described in paragraph 7.1.

The purpose of this section is to identify where the metric data was found, how it was collected during this study, and what automated tools are available that could be used to collect the data in future applications.

During this effort most metric data was collected manually. Some automated tools were available and used. Other tools were identified that would provide metric data. Table 8.1-1 provides summary information on the collection techniques used.

Table 8.1-1 Summary of Collection Techniques

DEVELOPMENT PHASES	REQUIREMENTS	DESIGN	IMPLEMENTATION
Number of elements	25	108	157
Collected during this study:			
Manually	25	100	144
Automatically	- (0%)	8 (7%)	13 (8%)
Can be automatically checked	6 (24%)	30 (28%)	83 (53%)

Thus, over 40% of the metric data can be collected via automated means. As more formal languages and techniques evolve for preparing requirements specifications and design specifications, a larger percentage of automated collection would be expected. The remaining metric data which is manually collected, in general, can be done quite routinely by trained personnel. There is the typical 10% of the data which requires a significant data collection effort. As experience with the metrics is gained, a determination of the significance of those metrics and whether they are worth the data collection effort required can be made.

Table 8.1-2 identifies the sources of the metric data and the number of elements which use that source for metric data, e.g., the system requirement specification is utilized as a source of data for 23 measurements (elements). More than one source may be used to arrive at a measurement, so the totals between this table and Table 8.1-1 do not directly correspond.

Table 8.1-2 Source Frequency

SOURCES	NUMBER OF TIMES USED AS SOURCE
Software System Requirements Specification	23
Standards and Conventions	8
Preliminary Design Specification	13
Preliminary Design Review Material	2
Detailed Design Specifications	98
Critical Design Review Material	2
Validation and Acceptance Test Specification	5
User's Manual/Operator's Manual	25
Interface Control Document	4
Data Base Management Plan	3
Problem Reports	2
Source Code	135
Training Material	2
Programmer's Notebook	2

SECTION 8

METRIC DATA COLLECTION

8.1 METRIC APPLICATION

While the historical data and sample size prevented the establishment of normalization functions for all of the quality factors, a significant benefit of the study was the experience gained in applying all of the metrics to the systems. Whether regression analysis was to be performed or not, the metric data was collected for all of the metrics.

It was found that applying the metrics to the various software products provided considerable insight into the design and implementation of the software. This reflects the n-tuple (indicator lights) concept described in paragraph 7.1.

The purpose of this section is to identify where the metric data was found, how it was collected during this study, and what automated tools are available that could be used to collect the data in future applications.

During this effort most metric data was collected manually. Some automated tools were available and used. Other tools were identified that would provide metric data. Table 8.1-1 provides summary information on the collection techniques used.

Table 8.1-1 Summary of Collection Techniques

DEVELOPMENT PHASES	REQUIREMENTS	DESIGN	IMPLEMENTATION
Number of elements	25	108	157
Collected during this study:			
Manually	25	100	144
Automatically	- (0%)	8 (7%)	13 (8%)
Can be automatically checked	6 (24%)	30 (28%)	83 (53%)

The frequency with which the sources are used indicates their importance to the resulting software end product and the quantifiability of their contents. For example, the source code is most frequently used and the detailed design specification is second most frequently used. The importance of the user's/operator's manual is indicated by its relatively high use. The relatively high use of the software system requirement specification and the preliminary design specification emphasizes the fact that some measures can ba applied very early in the development phase.

Appendix D provides a detailed examination of each measure, identifying where it was collected for this study and what type tool is available to automate its collection. The next two paragraphs briefly describe the automated tools used to collect the metric data during this study (paragraph 8.2) and those tools applicable to this task (paragraph 8.3) but not available during this study.

In Appendix D, examples are used to highlight the procedures and tools used to determine the measures. To illustrate the contents of this appendix, the following example is provided:

> An element of the design structure metric is based on the number of modules which do not have a single entrance and a single exit (SI.1(6)). During the design phase of a software development, an example of an automated tool which provides data for this measure is the Integrated Software Development System (GE/ISDS), described in paragraph 8.2. Using design charts in machine readable form, GE/ISDS performs various analyses on the design of individual programs. One such analysis identifies routines which have multiple entrances and exits. An example of a design chart and the resulting automated analysis is shown in Figure 8.1-1.

Appendix D contains many other such examples.

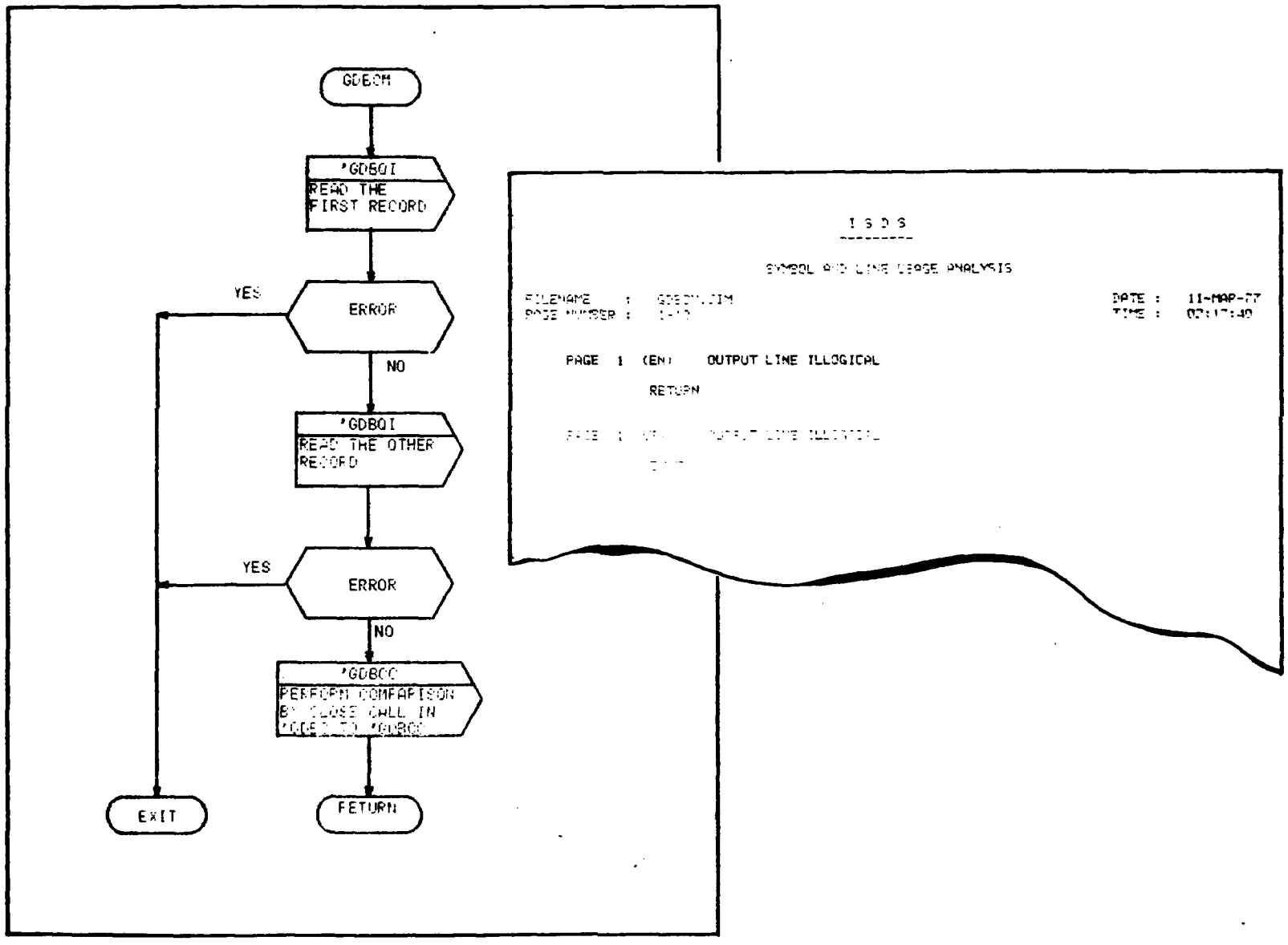


Figure 8.1-1 Automated Standards Checking During Design

8.2 TOOLS USED FOR DATA EXTRACTION

The following software support tools were used during this study to automatically collect metric data. Examples of outputs from the tools are in Appendix D corresponding to the measures which they provided.

A brief overview oriented toward describing the capabilities of each tool is provided. The intent in describing these tools and providing examples of their output in Appendix D is to emphasize that automated metric application is possible early in the development phase.

8.2.1 GE/INTEGRATED SOFTWARE DEVELOPMENT SYSTEM (GE/ISDS)

GE/ISDS is an integrated system of software support tools based on a common data base of software development information. Current capabilities emphasize analyses utilizing machine readable design charts. Some of the automated analyses include analysis of the design charts for compliance with standards, flow path, minimum number of tests required, and connectivity. Prototype versions of several other tools include methods for using structured programming constructs in flowcharts, interactive data base usage/structure definition, a measure of program complexity based on control structure and variable usage, and a formalized test procedure language for thorough testing of program segments ([CHANP76], [RICHP74], [RICHP76]).

8.2.2 CODE AUDIT ROUTINES (GJSUMRY/ATP)

These routines provide a profile of software characteristics for JOVIAL (J4) code. Included in the profile for each routine are counts of the number of cards, statements, procedures, declarations, comments, IFs, FORs, direct code statements, GOTOs, breaks from loops, operators, operands, delimiters, and other specified JOVIAL constructs ([ALGEC77]).

8.2.3 CONFIGURATION MANAGEMENT SYSTEM

As an aid to the strict configuration control of the development of software and any changes made, this system maintains a current status of any problem reports recorded against a routine. The system provides a log of any changes made and actions taken with regard to the software being developed.

8.2.4 REQUIREMENTS TRACE ROUTINE

This routine maintains a current list of performance requirements identified with itemized software system requirement specifications.

The impact of using these tools during this study was quite significant. Far less data collection would have been possible if all data collection had been done manually. The advantages in accuracy and manpower savings of automated data collection stresses its importance to the application of metrics.

8.3 OTHER TOOLS APPLICABLE TO METRIC DATA COLLECTION

Several other software support tools have been identified which appear to be applicable to metric data collection. A brief description of several will be provided in this section. The intent of this section is not to provide a support software tools survey, so the descriptions will be generic in nature. Examples of available tools will be mentioned.

8.3.1 REQUIREMENTS SPECIFICATION LANGUAGE/ANALYZER

The underlying concept of this tool is that if the requirements specification is written in a formal language, some form of analyses can be made on the specification. The analyses that can be done that relate to our metrics fall within the completeness and consistency areas. Examples of this tool are PSL/PSA [TIECD76] and RSL [BELLT76].

8.3.2 PROGRAM DESIGN LANGUAGE/ANALYZER

Consistent with the concept expressed above, if the design specification is written in a formal language (PDL), some analyses can be automatically performed. GE/ISDS provides these type of capabilities based on design charts. Planned enhancements are to provide the same analysis capabilities for a PDL. Some examples of work in this area are the PDL [PROG75] concept originated by IBM and the HOS concept [HAMIM76].

8.3.3 AUTOMATED VERIFICATION SYSTEM

These support software tools involve instrumenting source code to measure test effectiveness. The structure of the code, as well as path usage and time data, is analyzed. Some assistance in generating test data is provided. Examples of

tools in this group include FLOW [RICHP76], ANALYZER [NBS74], NODAL [NODA75], PET [PET72], and JAVS ([BROON76], [MILLE74]).

8.3.4 TEST PROCEDURE LANGUAGE

This tool is currently under development and is based on the use of a test procedure language (TPL) to formally state and document test procedures and a VERIFIER to apply the test procedures to the target modules or system. The test procedures are a deliverable product of the software development process and are used for both initial checkout and subsequent regression testing of target program modifications [PANZD76].

8.3.5 EXECUTION ANALYZER

Considerable information is gained by executing code under various loading conditions. A post execution routine could provide automated analysis and reports of pertinent metric data based on the execution. Such information as run time, core usage, module link-time and OS link-time are some of the examples indicated in Appendix D that could be reported via an automated tool.

8.3.6 CONSISTENCY CHECKER

This type tool provides the capability to identify various consistency measures relating to data, variable usage and initialization as well as others. Consistency checking can be done at the code level [RAMAC75] or at a specification level, such as extensions to PSL/PSA would provide.

8.3.7 DATA BASE ANALYZER

Analysis can be performed on the data base via tools such as a data definition language processor or a data base optimizer. The analyses center on data usage, data structure, and data redundancy.

These tools represent a sample of those available. The major concern would be of effectively using a subset of these tools in a software development environment. An integrated concept would be required.

REFERENCES

ABERD72 Abernathy, D.H., et al, "Survey of Design Goals for Operating Systems", Georgia Tech, GITIS-72-04, 1972.

ACQU71 "Acquisition and Use of Software Products for Automatic Data Processing Systems in the Federal Government", Comptroller General of the U.S., Report to the Congress, June 1971.

AIRF76 "Air Force Systems Command", Aviation Week & Space Technology, 19 July 1976.

ALGEC77 Algea, C., "ATP - Analysis of JOVIAL (J4) Routines", Internal GE Working Paper, March 1977.

AMORW73 Amory, W., Clapp, J.A., "An Error Classification Methodology", MITRE Tech Report, June 1973.

BELLD74 Bell, D.E., Sullivan, J.E., "Further Investigations into the Complexity of Software", MITRE Tech Report MTR-2874, June 1974.

BELLT76 Bell, T., et al, "An Extendable Approach to Computer-Aided Software Requirements Engineering", 1976 Software Engineering Conference.

BENSJ76 Benson, J., "Some Observations Concerning the Structure of FORTRAN Programs", International Symposium on Fault Tolerant Computing, Paris, June 1975.

BOEHB73a Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.S., Merritt, N.J., "Characteristics of Software Quality", Doc. #25201-6001-RU-00, NBS Contract #3-36012, 28 December 1973.

BOEHB76 Boehm, B., Brown, J., Lipow, M., "Quantitative Evaluation of Software Quality", 1976 Software Engineering Conference.

BOEHB73b Boehm, B.W., "Software and its Impact: A Quantitative Approach", Datamation, April 1973.

BOLEN76 Bolen, N., "An Air Force Guide to Contracting for Software Acquisition", NTIS AD-A020 444, January 1976.

BOULD61 Boulanger, D.G., "Program Evaluation and Review Technique", Advanced Management, July-August 1961.

BRADG75 Bradley, G.H., et al, "Structure and Error Detection in Computer Software", Naval Postgraduate School, NTIS AD-A014 334, February 1975.

BROON76 Brooks, N., et al, "Jovial Automated Verification System (JAVS)", RADC-TR-20, February 1976.

BROWJ73 Brown, J.R. and Buchanan, H.N., "The Quantitative Measurement of Software Safety and Reliability", TRW Report SS-73-06, August 1973.

BROWP72 Brown, P., "Levels of Language for Portable Software", Communications of the ACM, December 1972.

CASEJ74 Casey, J.K., "The Changing Role of the In-House Computer Application Software Shop", GE TIS #74AEG195, February 1974.

CHANP76 Chang, P., Richards, P.K., "Software Development and Implementation Aids", GE TIS #76CIS01, January 1976.

CHENL74 Cheng, L., Sullivan, J.E., "Case Studies in Software Design", MITRE Tech Report MTR-2874, June 1974.

CLAPJ74 Clapp, J.A., Sullivan, J.E., "Automated Monitoring of Software Quality", Proceedings from AFIPS Conference, Vol. 43, 1974.

COHEA72 Cohen, A., "Modular Programs: Defining the Module", Datamation, March 1972.

COMP69 "Computer Program Development and Configuration Management", AFSCF Exhibit 375-2, March 1969.

COMP66a "Computer Program Development and Configuration Management for the Manned Orbit Laboratory Program", SAFSL Exhibit 20012, September 1966.

COMP66b "Computer Program Subsystem Development Milestones", AFSCF SSD Exhibit 61-47B, April 1966.

CONF64 "Configuration Management During Definition and Acquisition Phases", AFSCM 375-1, June 1964.

CONF66 "Configuration Management of Computer Programs", ESD Exhibit EST-1, Section H, 1966.

CONNJ75 Connolly, J., "Software Acquisition Management Guidebook: Regulations, Specifications, and Standards", NTIS AD-A016 401, October 1975.

COOLW62 Cooley, T., Multivariate Procedures for the Behavioral Sciences, John Wiley and Sons, Inc., N.Y., 1962.

CORRA74 Corrigan, A.E., "Results of an Experiment in the Application of Software Quality Principles", MITRE Tech Report MTR-2874, June 1974.

CULPL75 Culpepper, L.M., "A System for Reliable Engineering Software", International Conference on Reliable Software, 1975.

DAVIC76 Davis, C., Vick, C., "The Software Development System", 1976 Software Engineering Conference.

DAVIR73 Davis, R.M., "Quality Software can Change the Computer Industry Programs Test Methods", Prentice-Hall, 1973, Chapter 23.

DENNJ70 Dennis, J.B., Goos, G., Poole, J., Gotlieb, C.C., et al, "Advanced Course on Software Engineering", Springer-Verlag, New York 1970.

DIJKE69a Dijkstra, E.W., "Complexity Controlled by Hierarchical Ordering of Function and Variability", Software Engineering, NATO Science Committee Report, January 1969.

DIJKE72 Dijkstra, E.W., "The Humble Programmer", Communications of the ACM, October 1972.

DIJKE69b Dijkstra, E.W., "Structured Programming", Software Engineering Techniques, NATO Science Committee Report, January 1969.

DIJKE72 Dijkstra, E.W., "Notes on Structured Programming", Structured Programming, Dahl, Dijkstra, Hoare, Academic Press, London 1972.

DOCU74 "Documentation Standards", Structured Programming Series Volume VII and Addendum, RADC-TR-74-300, September 1974 and April 1975.

DODM72 "DOD Manual for DOD Automated Data Systems Documentation Standards", DOD Manual 4120.17M, December 1972.

DROSM76 Drossman, M.M., "Development of a Nested Virtual Machine, Data Structure Oriented Software Design Methodology and Procedure for its Evaluation", USAFOSR/RADC Tech Report, 11 August 1976.

DUNSH77 Dunsmore, H., Ganon, J., "Experimental Investigation of Programming Complexity", Proceedings of ACM/NBS Sixteenth Annual Technical Symposium, June 1977.

EDWAN75 Edwards, N.P., "The Effect of Certain Modular Design Principles on Test-ability", International Conference on Reliable Software, 1975.

ELEC75 "The Electronic Air Force", Air Force Magazine, July 1975.

ELSHJ76 Elshoff, J.L., "Measuring Commercial PL/1 Programs Using Halstead's Criteria", SIGPLAN Notices, May 1976.

ELSHJ76b Elshoff, J., "An Analysis of Some Commercial PL/1 Programs", IEE Transactions on Software Engineering Volume SE-2, No. 2, June 1976.

ENDRA75 Endres, A., "An Analysis of Errors and their Causes in Systems Programs", International Conference on Reliable Software, 1975.

FAGAM76 Fagan, M., "Design and Code Inspections and Process Control in the Development of Programs", IBM TR 00.2763, June 1976.

FIND75 "Findings and Recommendations of the Joint Logistics Commanders", Software Reliability Working Group, November 1975.

FITZA76 Fitzsimmons, A., Love, T., "A Review and Critique of Halstead's Theory of Software Physics", GE TIS #76ISP004, December 1976.

FLEIJ72 Fleiss, J.E., et al, "Programming for Transferability", RADC-TR-72-234, September 1972.

FLEIT66 Fleishman, T., "Current Results from the Analysis of Cost Data for Computer Programming", NTIS AD-637 801, August 1966.

GILBT76 Gilb, T., Software Metrics, Winthrop Computer Systems Series, 1976.

GOODJ74 Goodenough, J., "Effect of Software Structure on Software Reliability, Modifiability, and Reusability: A Case Study", USA Armament Command, March 1974.

GOODJ75 Goodenough, J., "Exception Handling Design Issues", SIGPLAN Notices, July 1975.

GOVE74 "Government/Industry Software Sizing and Costing Workshop-Summary Notes", USAFESD, 1-2 October 1974.

HAGAS75 Hagan, S., "An Air Force Guide for Monitoring and Reporting Software Development Status", NTIS AD-A016 488, September 1975.

HAGUS76 Hague, S.J., Ford, B., "Portability-Prediction and Correction", Software Practices & Experience, Vol. 6, 61-69, 1976.

HALSM77 Halstead, M., Elements of Software Science, Elsevier Computer Science Library, N.Y., 1977.

HALSM73 Halstead, M., "Algorithm Dynamics", Proceedings of Annual Conference of ACM, 1973.

HALSM72 Halstead, M., "Natural Laws Controlling Algorithm Structure", ACM SIGPLAN, February 1972.

HAMIM76 Hamilton, M., Zeldin, S., "Integrated Software Development System/Higher Order Software Conceptual Description", ECOM-76-0329-F, November 1976.

HANEF72 Haney, F.M., "Module Connection Analysis - A Tool for Scheduling Software Debugging Activities", Proceedings of the 1972 Fall Joint Computer Conference, Vol. 41, Part 1, 173-179, 1972.

HODGB76 Hodges, B., Ryan, J., "A System for Automatic Software Evaluation", 1976 Software Engineering Conference.

JONEC77 Jones, C., "Program Quality and Programmer Productivity", IBM TR 02.764, January 1977.

KERNB74 Kernighan, B., Plauger, P., The Elements of Programming Style, McGraw-Hill, 1974.

KESSM70 Kessler, M.M., "An Investigation of Program Structure", IBM Federal Systems Division, Internal Memo, February 1970.

KNUTD68 Knuth, D.E., The Art of Computer Programming Vol. 1, Addison-Wesley, 1968.

KNUTD71 Knuth, D.E., "An Empirical Study of FORTRAN Programs", Software Practice & Experience, Vol. 1, pp 105-133, 1971.

KOSAS74 Kosarajo, S.R., Ledgard, H.F., "Concepts in Quality Software Design", NBS Technical Note 842, August 1974.

KOSYD74 Kosy, D., "Air Force Command and Control Information Processing in the 1980s: Trends in Software Technology", Rand, June 1974.

KUESJ73 Keuster, J., Mize, J., Optimization Techniques with FORTRAN, McGraw-Hill, N.Y., 1973.

LABOV66 LaBolle, V., "Development of Equations for Estimating the Costs of Computer Program Production", NTIS AD-637 760, June 1966.

LAPAL73 LaPadula, L.J., "Software Reliability Modeling and Measurement Techniques", MTR-2648, June 1973.

LARSR75 Larson, R., "Test Plan and Test Case Inspection Specification", IBM TR 21.586, April 1975.

LEWIE63 Lewis, E., Methods of Statistical Analysis, Houghton Mifflin Company, Boston 1963.

LIEBE72 Lieblein, E., "Computer Software: Problems and Possible Solutions", CENTACS USAECOM Memorandum, 7 November 1972.

LIGHW76 Light, W., "Software Reliability/Quality Assurance Practices", Briefing given at AIAA Software Management Conferences, 1976.

LISKB75 Liskov, B., "Data Types and Program Correctness", SIGPLAN Notices, July 1975.

LISKB73 Liskov, B.H., "Guidelines for the Design and Implementation of Reliable Software Systems", MITRE Report 2345, February 1973.

LOVET76a Love, T., Bowman, A., "An Independent Test of the Theory of Software Physics", SIGPLAN Notices, November 1976.

LOVET76b Love, T., Fitzsimmons, A., "A Survey of Software Practioners to Identify Critical Factors in the Software Development Process", GE TIS 76ISP003, December 1976.

MANNJ75 Manna, J., "Logical Analysis of Programs", International Conference on Reliable Software, 1975.

MARSS70 Marshall, S., Millstein, R.E., Sattley, K., "On Program Transferability", Applied Data Research, Inc., RADC-TR-70-217, November 1970.

MCCAT76 McCabe, T., "A Complexity Measure", 1976 Software Engineering Conference.

MCCRD72 McCracken, D.D. and Weinberg, G.M., "How to Write a Readable FORTRAN Program", Datamation, October 1972.

MCKIJ77 McKissick, J., Price, R., "Quality Control of Computer Software", 1977 ASQC Technical Conference Transactions, Philadelphia 1977.

MCNEL75 McNeely, L., "An Approach to the Development of Methods and Measures for Quantitatively Determining the Reliability of Software", Ultra Systems Concept Paper, February 1975.

MEALG68 Mealy, G.H., Farber, D.J., Morehoff, E.E., Sattley, "Program Transferability Study", RADC, November 1968.

MILI70 "Military Standard Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs", MIL-STD-483, December 1970.

MILI68 "Military Standard Specification Practices", MIL-STD-490, October 1968.

MILLE74 Miller, E., et al, "JOVIAL/J3 Automated Verification System (JAVS) System Design Document", GRC, March 1974.

MULOR70 Mulock, R.B., "A Study of Software Reliability at the Stanford Linear Accelerator Center, Stanford University", August 1970.

MYERG73 Myers, G.J., "Characteristics of Composite Design", Datamation, September 1973.

MYERG75 Myers, G.J., Reliable Software through Composite Design, Petrocelli/Charter, 1975.

MYERG76 Myers, G.J., Software Reliability: Principles and Practices, John Wiley & Sons, New York, 1976.

NBS74 "Analyzer - Computation and Flow Analysis", NBS Tech Note 849, 1974.

NELSR74 Nelson, Richard, "A Plan for Quality Software Production", RADC Internal Paper, June 1974.

NELSR75 Nelson, R., Sukert, A., "RADC Software Data Acquisition Program", RADC Paper presented at Fault Tolerant System Workshop, Research Triangle Institute, November 1975.

NODA75 "NODAL - Automated Verification System", Aerospace TOR-0075(5112)-1, 1975.

OGDIJ72 Ogdin, J.L., "Designing Reliable Software", Datamation, July 1972.

OSTEL74 Osterweil, L., et al, "Data Flow Analysis as an Aid in Documentation, Assertion Generation, and Error Detection", NTIS PB-236-654, September 1974.

OSTLB63 Ostle, B., Statistics in Research, Iowa State University Press, 1963.

PADED56 Paden, D., Linquist, E., Statistics for Economics and Business, McGraw-Hill, New York, 1956.

PANZD76 Panzl, D., "Test Procedures: A New Approach to Software Verification", 1976 Software Engineering Conference.

PARIR76 Pariseav, R., "Improved Software Productivity for Military Systems through Structured Programming", NTIS AD-A022 284, March 1976.

PARND72a Parnas, D.L., "A Technique for Software Module Specification with Examples", Communications of the ACM, Vol. 15 No. 5, 1972.

PARND71 Parnas, D.L., "Information Distribution Aspects of Design Methodology", Proc IFIP Congress 1971.

PARND75 Parnas, D.L., "The Influence of Software Structure on Reliability", International Conference on Reliable Software, 1975.

PARND72b Parnas, D.L., "On the Criteria to be used in Decomposing Systems into Modules", Comm. of the ACM, Vol. 15, No. 12, December 1972.

PATH76 Pathway Program - Product Quality Assurance for Shipboard Installed Computer Programs, Naval Sea Systems Command, April 1976.

PET72 "PET - Automatic Test Tool", AFIPS Conference Proceedings, Vol. 42, 1972.

PILIM68 Piligian, M.S., et al, "Configuration Management of Computer Program Contract End Items", ESD-TR-68-107, January 1968.

POOLL77 Poole, L., Borchers, M., Some Common Basic Programs, Adam Osborne and Associates, Berkeley, 1977.

PROG75 Program Design Study "Structured Programming Series" (Vol. VIII), RADC TR-74-300, 1975.

RAMAC75 Ramamoorthy, C., Ho, S., "Testing Large Software with Automated Software Evaluation Systems", 1976 Software Engineering Conference.

REIFD75 Reifer, D.J., "Automated Aids for Reliable Software", International Conference on Reliable Software, 1975.

REIFD76 Reifer, D., "Toward Specifying Software Properties", IFIP Working Conference on Modeling of Environmental Systems, Tokyo, Japan, April 1976.

RICHF74 Richards, F.R., "Computer Software Testing, Reliability Models, and Quality Assessment", NTIS AD-A001 260, July 1974.

RICHP74 Richards, P., et al, "Simulation Data Processing Study: Language and Operating System Selection", GE TIS 74CIS09, June 1974.

RICHP75 Richards, P., Chang, P., "Software Development and Implementation Aids IR&D Project Final Report for 1974", GE TIS 75CIS01, July 1975.

RICHP76 Richards, P., Chang, P., "Localization of Variables: A Measure of Complexity", GE TIS 76CIS07, December 1976.

ROSED76 Rosenkrantz, D., "Plan for RDL: A Specification Language Generating System", GE Internal Document, March 1975.

RUBER68 Rubey, R.J., Hartwick, R.D., "Quantitative Measurement of Program Quality", Proceedings of 23rd National Conference, ACM, 1968.

SABIM76 Sabin, M.A., "Portability - Some Experiences with FORTRAN", Software-Practice & Experience, Vol. 6, pp 393-396, 1976.

SACI76 "SAC in Transition", Aviation Week and Space Technology, 10 May 1976.

SACKH67 Sackman, H., Computers, System Science, and Evolving Society, J. Wiley & Sons, 1967.

SALIJ77 Salinger, J., "Initial Report on the Feasibility of Developing a Work Measurement Program for the Data Processing Departments", Blue Cross/ Blue Shield Internal Paper, January 1977.

SALVA75 Salvador, A., Gordon, J., Capstick, C., "Static Profile of Cobol Programs", SIGPLAN Notices, August 1975.

SAMS75 "SAMSO Program Management Plan Computer Program Test and Evaluation", February 1975.

SCHNN72 Schneidewind, N.F., "A Methodology for Software Reliability Prediction and *Quality Control*", *Naval Postgraduate* School, NTIS AD-754 377, November 1972.

SCHNN75 Schneidewind, N.F., "Analysis of Error Processes in Computer Software", International Conference on Reliable Software, 1975.

SCHOJ76 Schonfelder, J.L., "The Production of Special Function Routines for a Multi-Machine Library", Software-Practice and Experience, Vol. 6, pp 71-82, 1976.

SCHOW76 Schoeffel, W., "An Air Force Guide to Software Documentation Requirements", NTIS AD-A027 051, June 1976.

SHOOM75a Shooman, M.L., Bolskey, M.I., "Software Errors: Types, Distribution, Test and Correction Times", International Conference on Reliable Software, 1975.

SHOOM75b Shooman, M., "Summary of Technical Progress - Software Modeling Studies", RADC Interim Report, September 1975.

SMITR74 Smith, R.,"Management Data Collection and Reporting - Structured Programming Series (Vol. IX)" RADC TR-74-300, October 1974.

SOFT75 "Software Engineering Handbook", GE Special Purpose Computer Center, September 1975.

SPAC76 "GE Space Division Task Force on Software", Engineering and Management June 28 Report, 1976.

STEWD74 Steward, D.W., "The Analysis of the Structure of Systems", GE TIS 74NED36, June 1974.

SULLJ73 Sullivan, J.E., "Measuring the Complexity of Computer Software", MITRE Tech Report MTR-2648, June 1973.

SUPP73 "Support of Air Force Automatic Data Processing Requirements through the 1980's", SADPR-85, July 1973.

SZABS76 Szabo, S., "A Schema for Producing Reliable Software", International Symposium on Fault Tolerant Computing, Paris, June 1975.

TACT74 "Tactical Digital Systems Documentation Standards", Department of the Navy, SECNAVINST 3560.1, August 1974.

TALIW71 Taliaferro, W.M., "Modularity: The Key to System Growth Potential", Software Practices and Experience, July-September 1971.

TEICD76 Teichroew, D., "PSL/PSA A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", 1976 Software Engineering Conference.

THAYT76 Thayer, T.A., Hetrick, W.L., Lipow, M., Craig, G.R., "Software Reliability Study", RADC TR-76-238, August 1976.

THAYT75 Thayer, T.A., "Understanding Software through Empirical Reliability Analysis", Proceedings, 1975 National Computer Conference.

USAR75 "US Army Integrated Software Research and Development Program", USACSC, January 1975.

VANDG74 VanderBrug, G.J., "On Structured Programming and Problem-Reduction", NSF TR-291, January 1974 (MF).

VANTD74 Van Tassel, Dennie, Program Style, Design, Efficiency, Debugging and Testing, Prentice-Hall, Inc., New Jersey, 1974.

VOLKW58 Volk, W., Applied Statistics for Engineers, McGraw-Hill Book Co., Inc., New York, 1958.

WALTG74 Walters, G.F., et al, "Spacecraft On-Board Processor/Software Assessment", GE TIS 74CIS10, June 1974.

WALTG76 Walters, G.F., "Software Aids Index", GE Internal Working Paper, December 1976.

WAGOW73 Wagoner, W.L., "The Final Report on a Software Reliability Measurement Study", Aerospace Report TOR-0074, August 1973.

WEING71 Weinberg, G.M., "The Psychology of Computer Programming", NY, Van Nostrand Reinhold, 1971.

WHIPL75 Whipple, L., "AFAL Operational Software Concept Development Program", Briefing given at Software Subpanel, Joint Deputies for Laboratories Committee, 12 February 1975.

WILLN76 Willmouth, N., "Software Data Collection: Problems of Software Data Collection", RADC Interim Report, 1976.

WOLVR72 Wolverton, R.W., Schick, G.J., "Assessment of Software Reliability", TRW Report SS-72-04, September 1972.

WULFW73 Wulf, W.A., "Report of Workshop 3 - Programming Methodology", Proceedings of a Symposium on the High Cost of Software, September 1973.

YOURE75 Yourdon, E., Techniques of Program Structure and Design, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

ZAHNC75 Zahn, C., "Structured Control in Programming Languages", SIGPLAN Notices, July 1975.

APPENDIX C

RESULTS OF DEVELOPMENT AND VALIDATION OF NORMALIZATION FUNCTIONS

This appendix is organized as follows:

For each quality factor:

- Analysis was not performed on certain metrics. These metrics are identified and reasons why they were not used is given according to the codes described in Table C-1.

- Regression analysis was performed on certain individual metrics. A summary of the metric scores and the quality factor ratings for the quality factor ratings for the subset of modules (System B) used is given as well as the results of the analysis: predictor coefficient, standard error of estimate, and correlation coefficient. The results are plotted on graphs.

- Based on these results, a second subset of modules (from System A) were plotted on the same graphs as a validation of the normalization functions developed in the above step. A summary of the rating and metric scores for this second subset is given for comparison. Acceptance or rejection of the normalization function is indicated.

- Based on these results, certain metrics were chosen to be used in the multiple regression. The results of this analysis are provided.

Table C-1 Reasons for No Analysis or Correlation

CODE	EXPLANATION
R1	NOT SUFFICIENT VARIATION IN DATA BASE - Very strict standards or a restriction of the development environment may cause a very limited range of metric scores which would limit statistical significance.
R2	NO HISTORICAL DATA AVAILABLE - It may be impossible to derive a rating of a quality factor because supporting data was not collected or the system had not experienced the activity represented by the quality factor. For example, if a system had not been moved from one environment to another, there would be no data to derive a rating of portability.
R3	DATA BASE DOES NOT SUPPORT METRIC DATA COLLECTION - Data may not be available in the data base which is required to determine a metric. For example, programmer notebooks which contained module level testing information were not available, therefore many of the instrumentation measures could not be applied. There was no source available for those metrics.
R4	SYSTEM LEVEL METRIC - Since only two systems were used, a larger sample is required.

CORRECTNESS

Regression analysis not performed on the following metrics.

Reasons: R1, R4

Table C-2 Data Collection Summary for Correctness

SYSTEM A & B		RQMTS		DESIGN			IMPLEMENTATION			
		CP.1	TR.1	CP.1	CS.1	CS.2	TR.1	CP.1	CS.1	CS.2
INDIVIDUAL METRIC ANALYSIS	AVERAGE	1	1	.98	.99	.8	1	.92	1	1
	RANGE	-	-	.87-1.	.75-1.	-	-	.75-1.	-	-
	STD DEV	-	-	.001	.06	-	-	.007	-	-

RELIABILITY

Regression analysis not performed on following metrics.

Reasons: R1, R4

Table C-3 Data Collection Summary for Reliability

SYSTEM A & B		RQMTS					DESIGN			
		AC.1	ET.2	ET.3	ET.4	ET.5	CS.1	CS.2	AC.1	ET.4
INDIVIDUAL METRIC ANALYSIS	AVERAGE	1	1	1	0	1	.99	.8	1	0
	RANGE	-	-	-	-	-	.75-1.	-	-	-
	STD DEV	-	-	-	-	-	.06	-	-	-

RELIABILITY (Continued)

Regression analysis not performed on following metrics.

Table C-3 Data Collection Summary for Reliability (Continued)

SYSTEM A & B		IMPLEMENTATION				
		CS.1	CS.2	AC.1	ET.4	SI.2
INDIVIDUAL METRIC ANALYSIS	AVERAGE	1	1	1	0	1
	RANGE	-	-	-	-	-
	STD DEV	-	-	-	-	-

RELIABILITY (Continued)

C-8

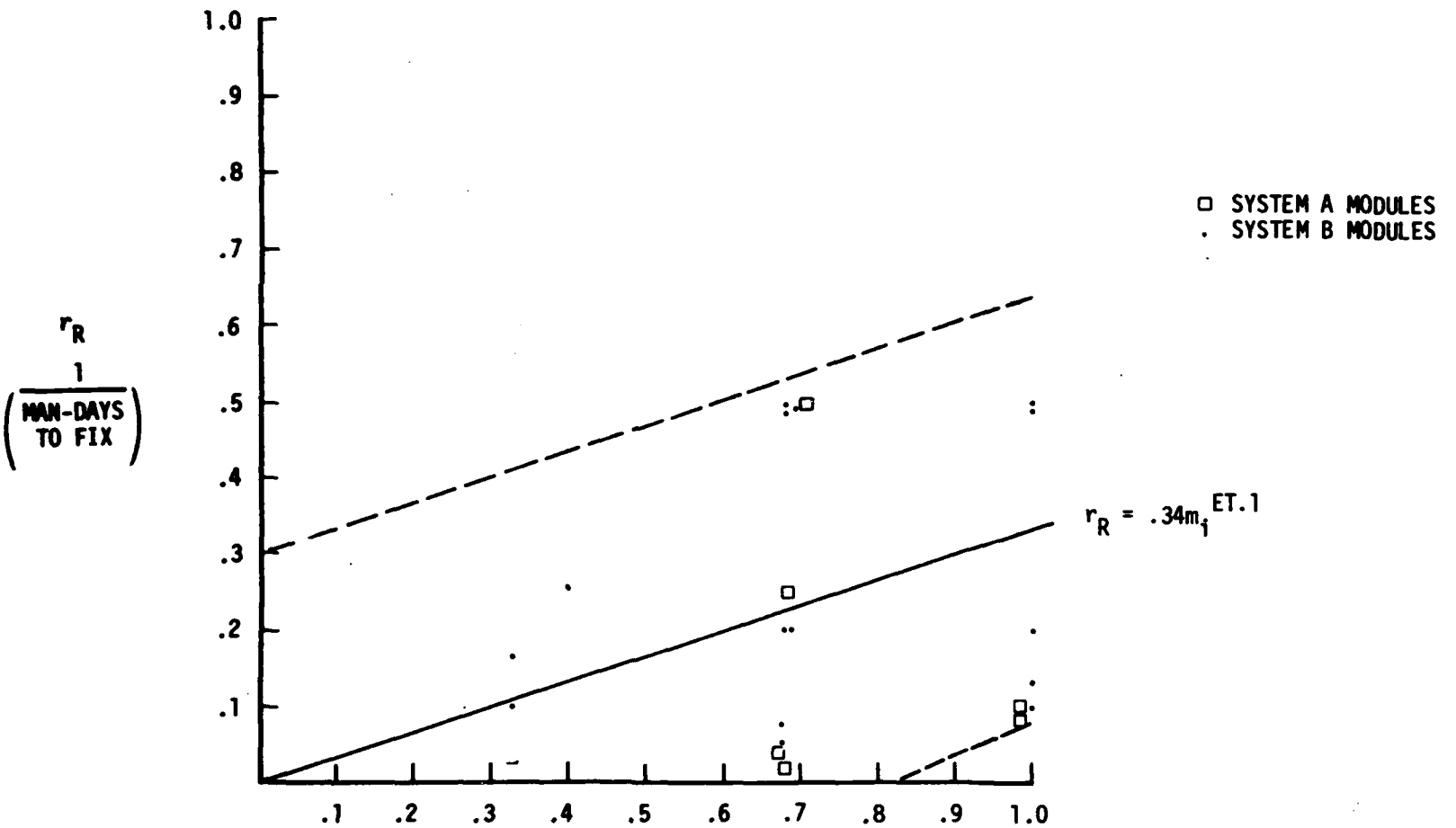
Regression analysis performed on following metrics.

Table C-4 Regression Analysis Summary for Reliability

SYSTEM B	DESIGN						RATING
	ET.1	ET.2	ET.3	ET.5	SI.1	SI.3	
Average	.77	.84	.51	.67	.702	.675	.235
Range	.33-1.	.5-1.	0-1.	0-1.	.29-.9	0-.99	.03-.5
Std Dev	.23	.20	.22	.47	.21	.35	.186
Predictor Coefficients	.34	.20		.25	.13	.34	
Std Error of Estimate	.18	.11		.17	.08	.16	
Correlation Coefficient	.82	.84		.63	.75	.85	

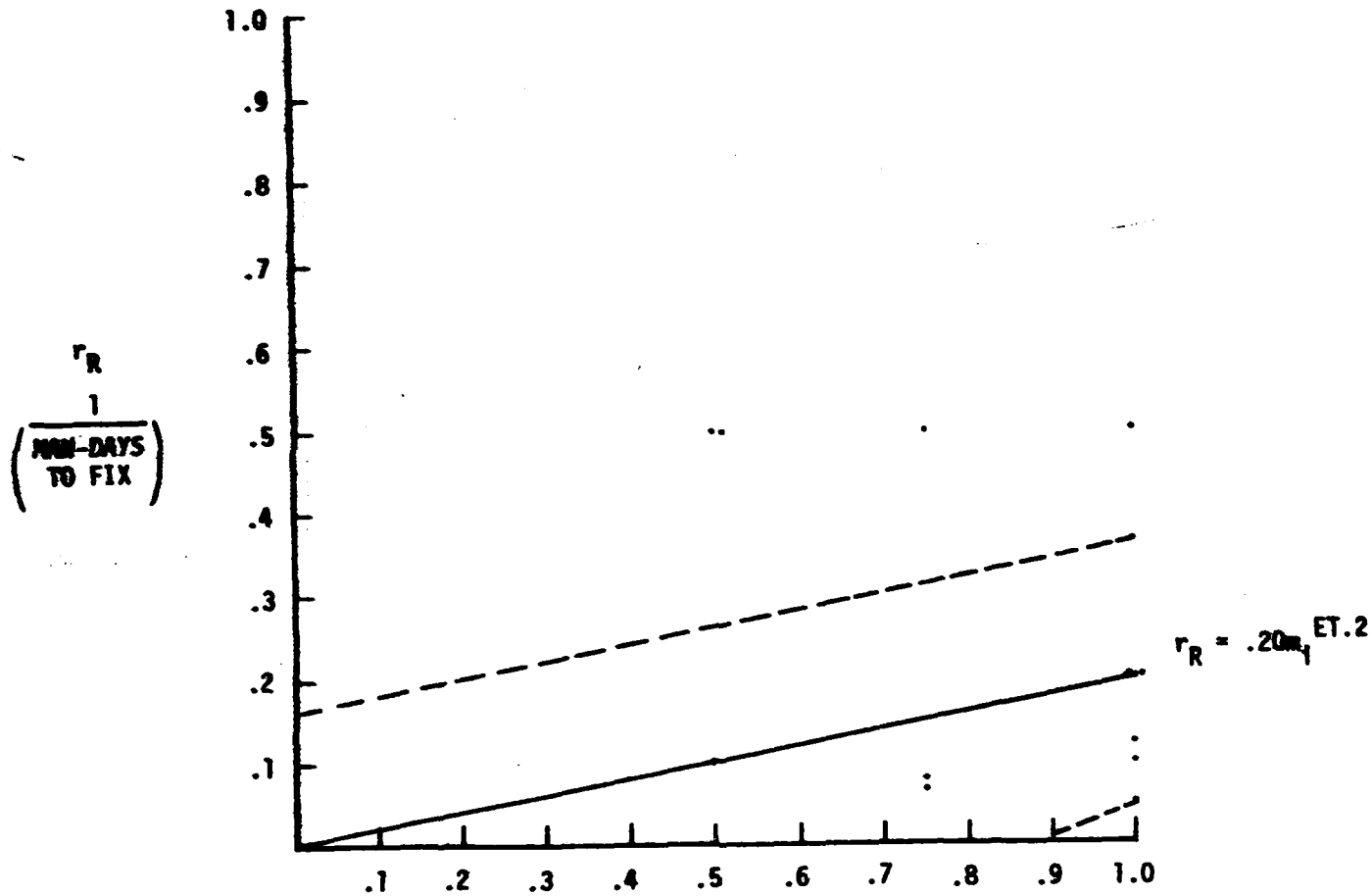
SYSTEM A	**	*	*	*	*	**	RATING
Average	.78					.61	.203
Range	.33-1.					0-.99	0-1.
Std Dev	.22					.40	.30
Multiple Regression	✓					✓	Included in set
Predictor Coefficients	.18					.19	
Std Error	.17						
Correlation Coefficient	.87						

* Rejected from further analysis, insignificant correlation.
** Normalization function accepted.



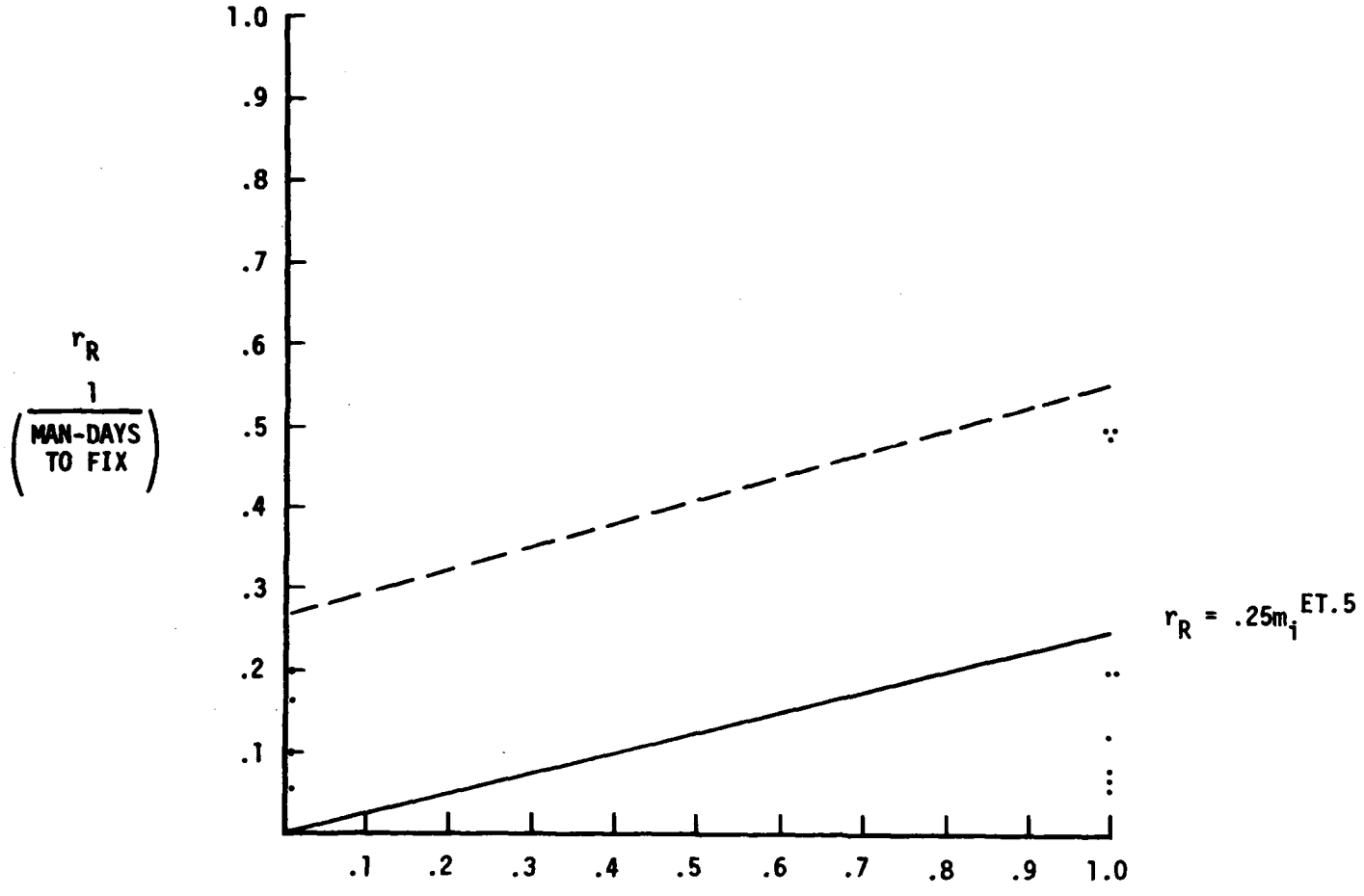
1608A

Figure C-1 $ET.1_R$ (Design) Normalization Function



16088

Figure C-2 $ET.2_R$ (Design) Normalization Function



1608C

Figure C-3 $ET.5_R$ (Design and Implementation) Normalization Function

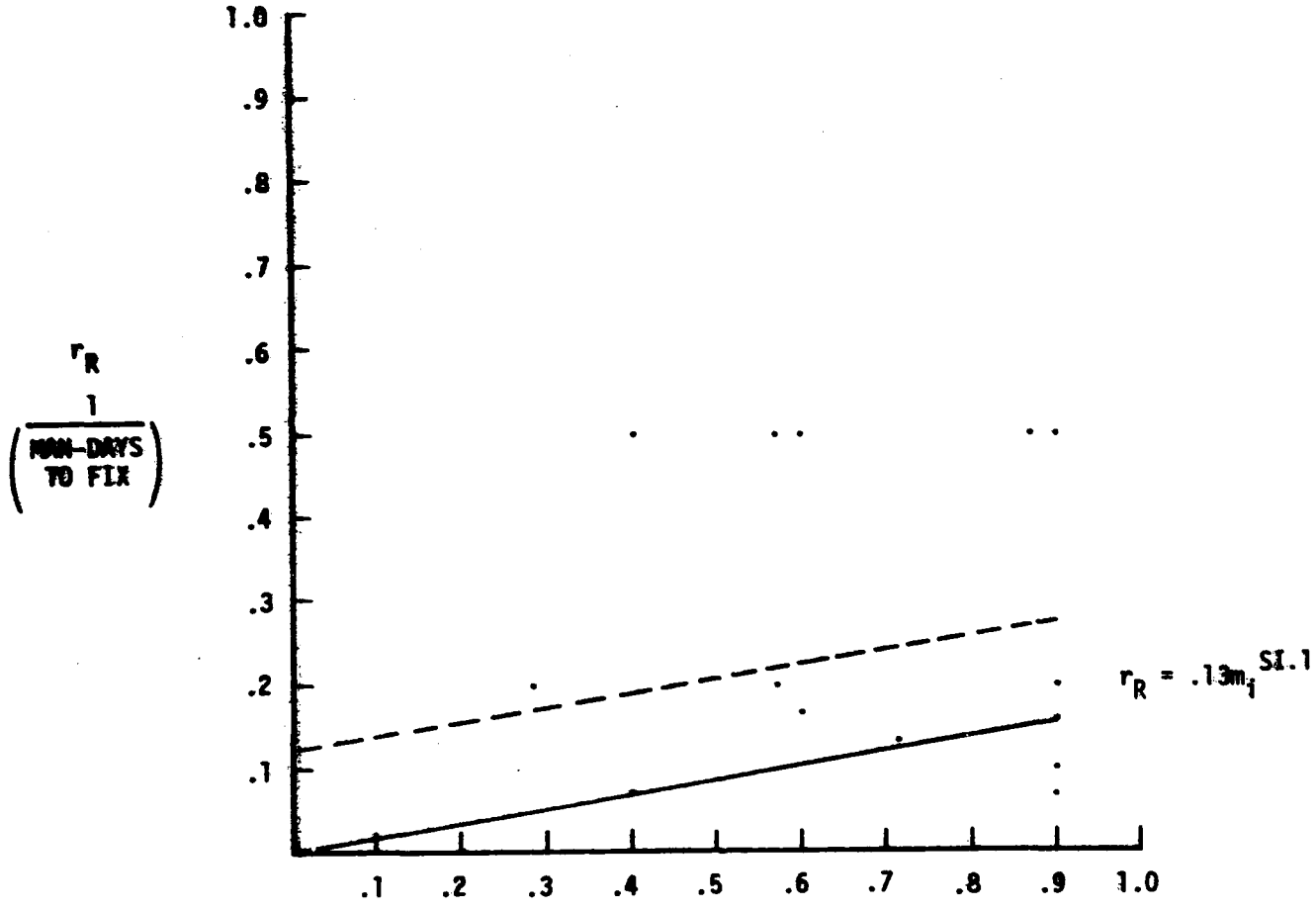
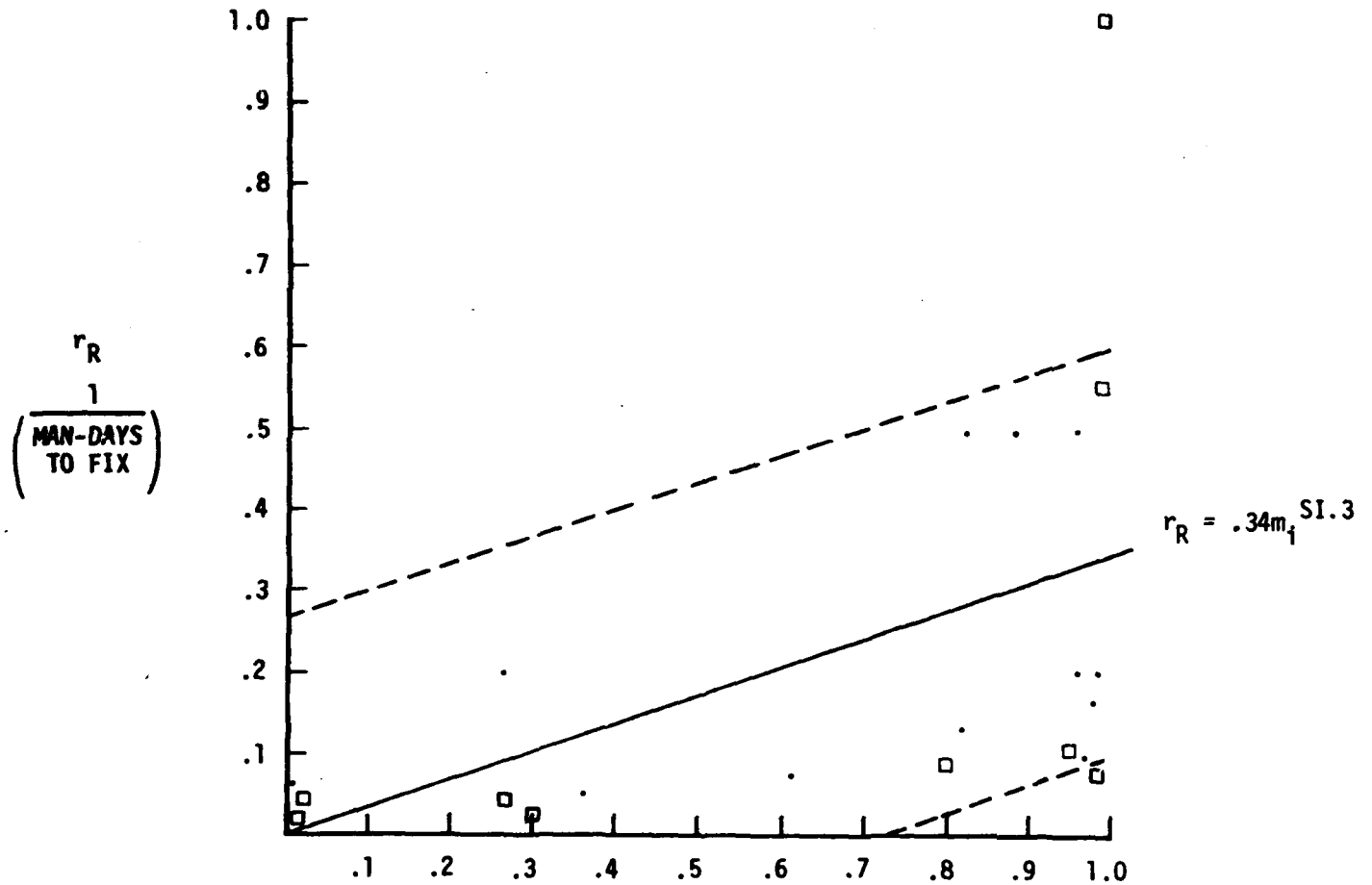


Figure C-4 $SI.1_R$ (Design) Normalization Function



1608E

Figure C-5 $SI.3_R$ (Design) Normalization Function

RELIABILITY (Continued)

Regression analysis performed on following metrics.

Table C-4 Regression Analysis Summary for Reliability (Continued)

SYSTEM B	IMPLEMENTATION							RATING
	ET.1	ET.2	ET.3	ET.5	SI.1	SI.3	SI.4	
Average	.79			.67	.64	.66	.76	.32
Range	.33-1.			0-1.	.17-.9	0-.99	.5-.9	.03-1.
Std Dev	.43			.47	.22	.36	.10	.15
Predictor Coefficients	.57	-	-	.25	.58	.53	.53	
Std Error of Estimate	.31			.17	.31	.32	.34	
Correlation Coefficient	.83			.63	.78	.78	.77	

SYSTEM A	*	*	*	*	**	**	**	RATING
Average	.79				.74	.61	.78	.28
Range	.33-1.				.17-1.	0-.99	.7-.9	0-1.
Std Dev	.22				.22	.40	.05	.39
Multiple Regression	✓				✓	✓	✓	Included in set
Predictor Coefficients	.48				.14	0	0	
Standard Error	.33							
Correlation Coefficient	.85							

* Rejected from further analysis, insignificant correlation.
** Normalization function accepted.

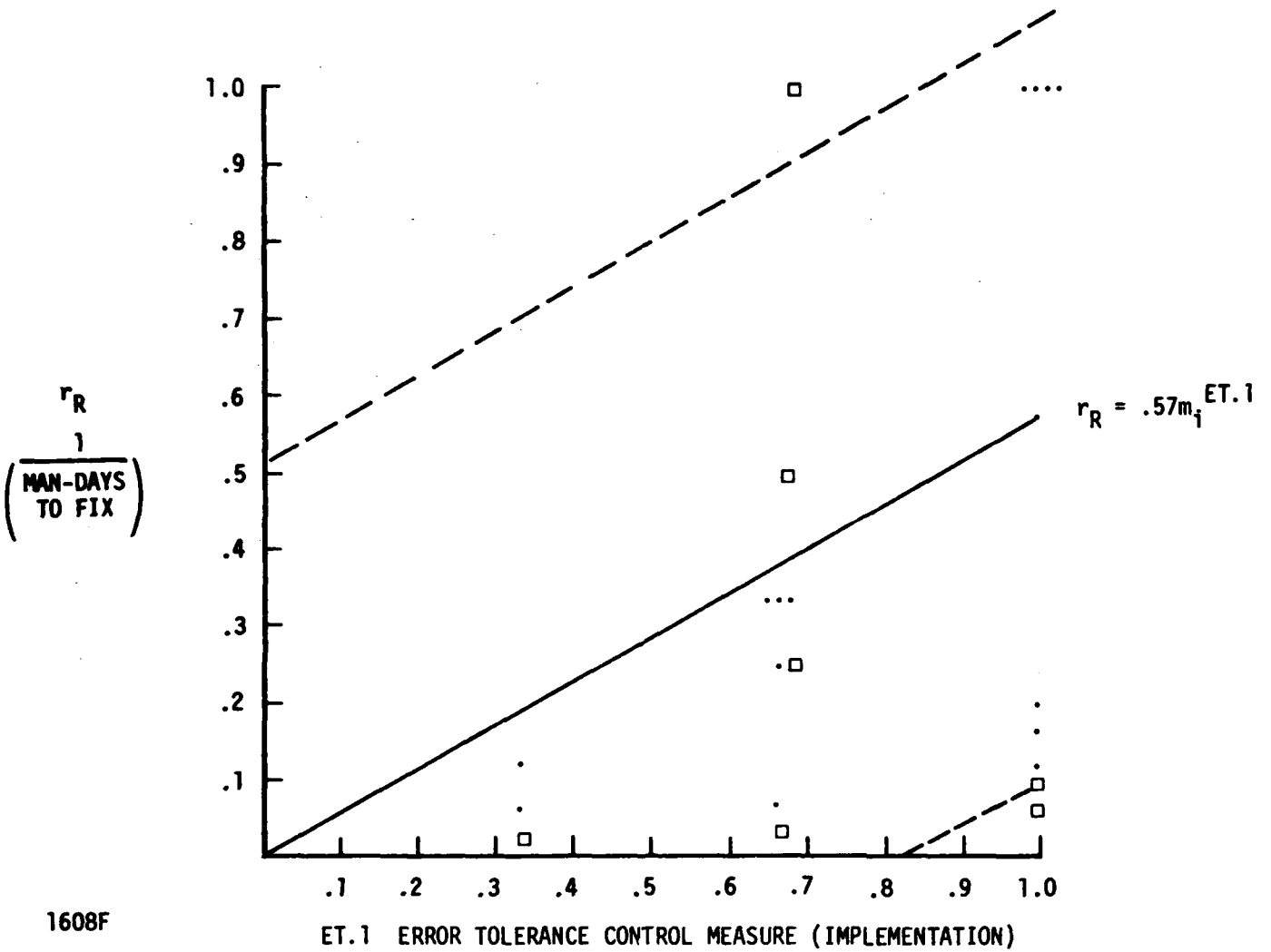
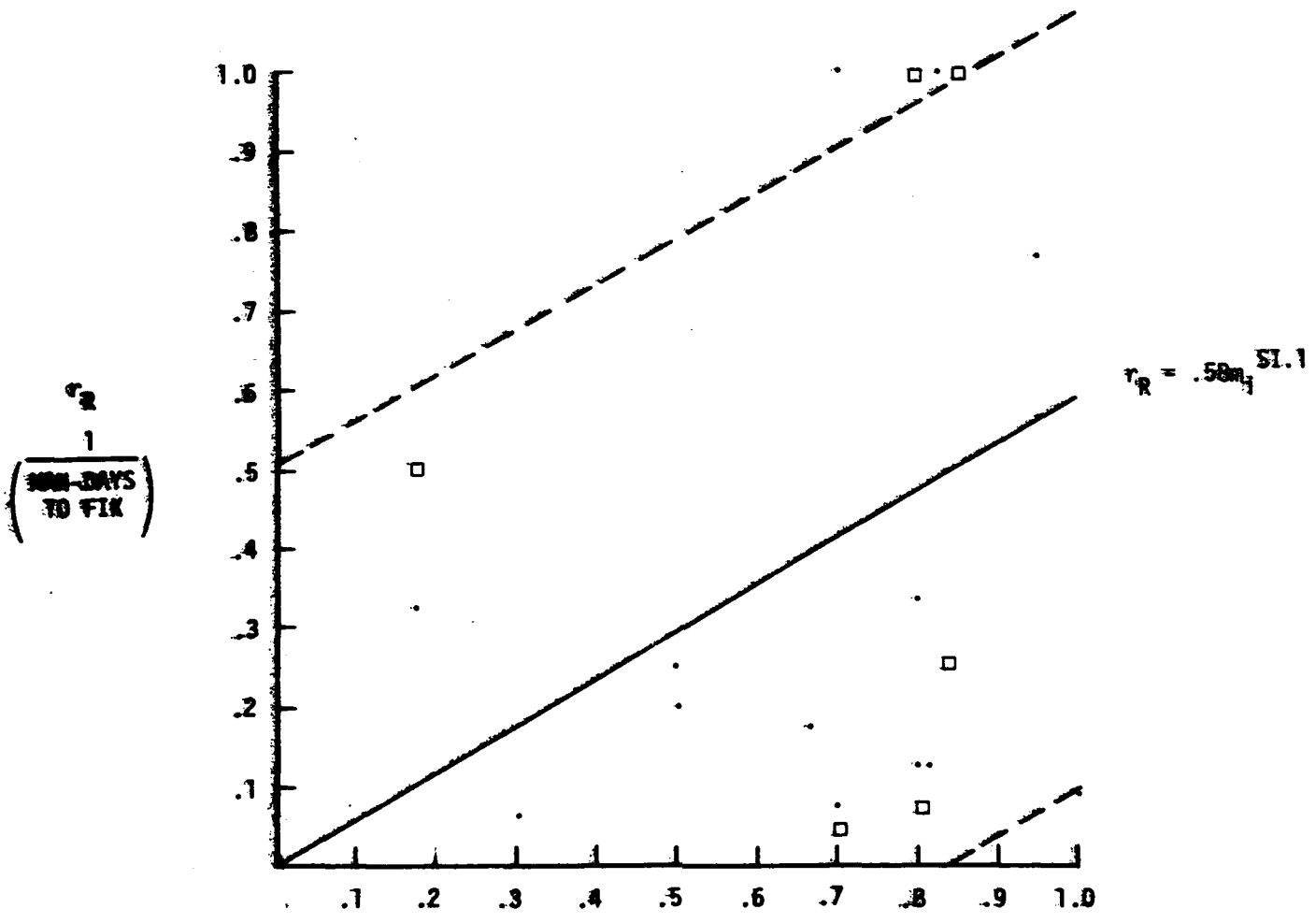


Figure C-6 $ET.1_R$ (Implementation) Normalization Function



1608G

Figure C-7 $SI.1_R$ (Implementation) Normalization Function

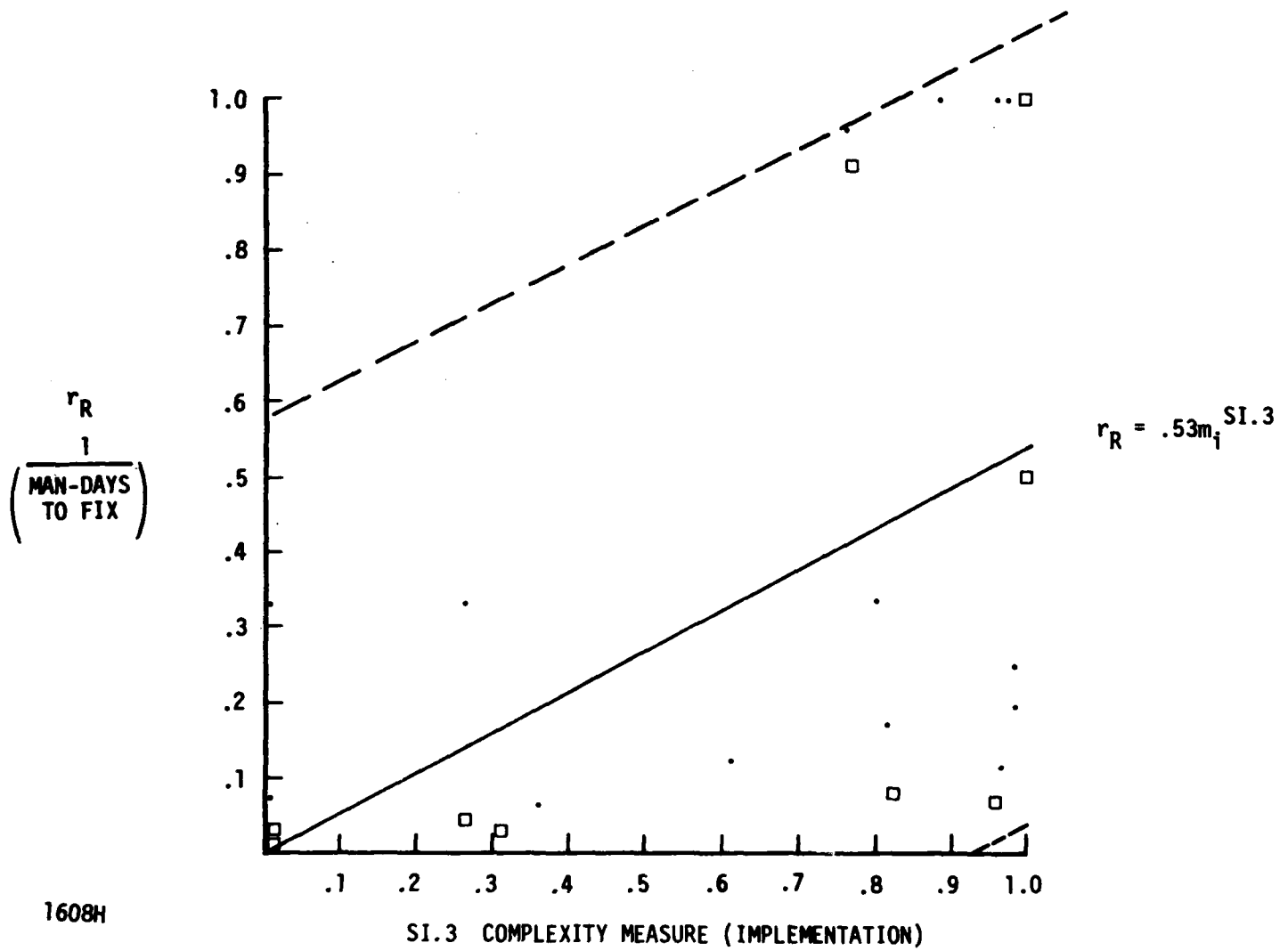


Figure C-8 $SI.3_R$ (Implementation) Normalization Function

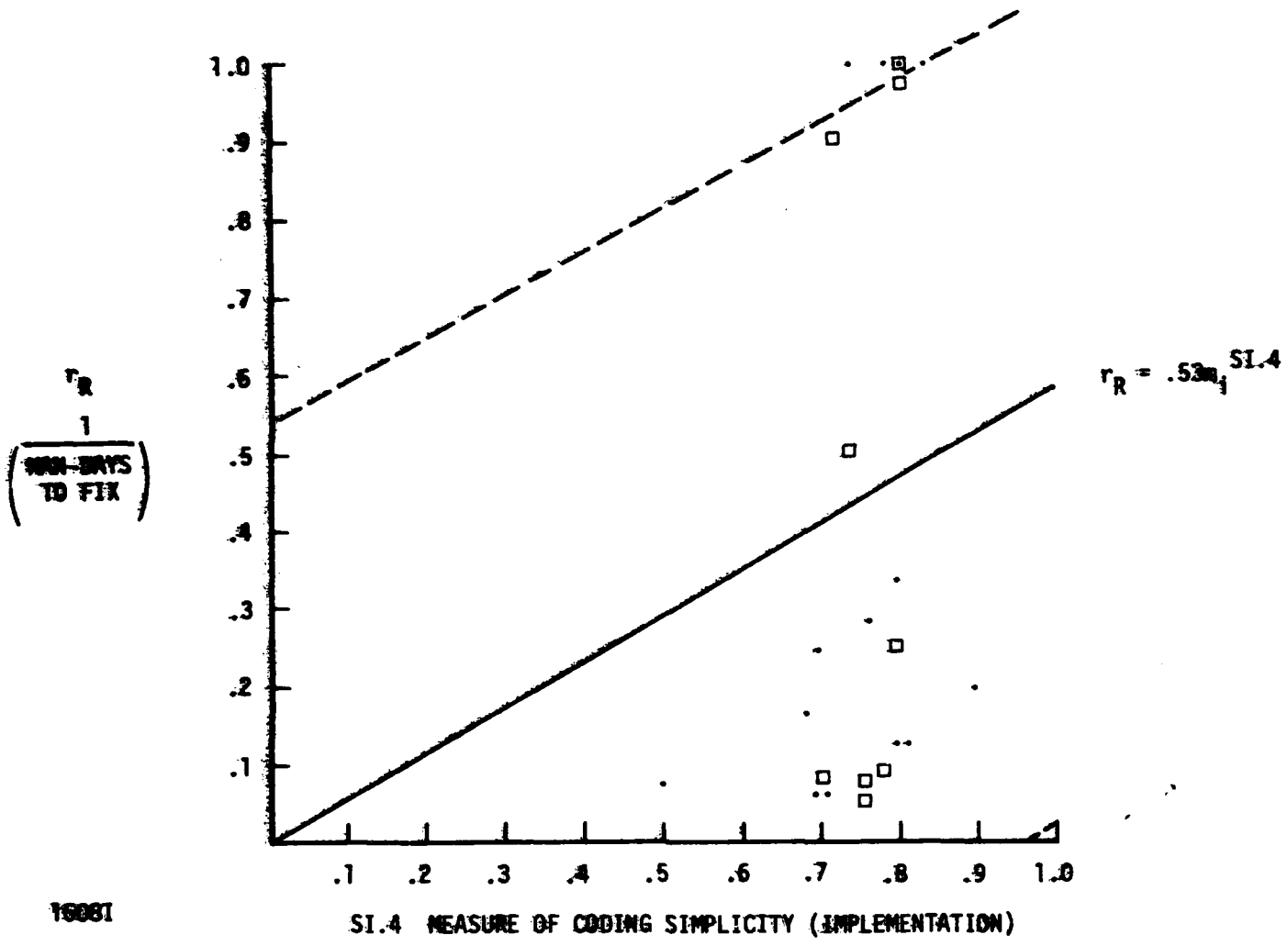


Figure C-9 $SI.4_R$ (Implementation) Normalization Function

EFFICIENCY

Regression analysis not performed.

Reasons: R2, R3

Table C-5 Data Collection Summary for Efficiency

SYSTEM A & B		DESIGN				IMPLEMENTATION		
		EE.1	EE.2	EE.3	SE.1	EE.2	EE.3	SE.1
INDIVIDUAL METRIC ANALYSIS	AVERAGE	1	.67	1	.5	.70	.96	.6
	RANGE	-	0-1.	-	-	.4-1.	.8-1	.4-.8
	STD DEV	-	.31	-	-	.13	.08	.06

INTEGRITY

Regression Analysis Not Performed

Reasons: R2, R4

Metric sources for AC.1 and AA.1 for both system A and B were zero since no access control or audit provisions are incorporated in software. Physical security is utilized.

USABILITY

Regression analysis not performed.

Reasons: R4

Table C-6 Data Collection Summary for Usability

SYSTEM A & B		RQMTS			DESIGN			
		OP.1	CM.1	CM.2	OP.1	TG.1	CM.1	CM.2
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.67	1	.5	.8	1	.74	.83
	RANGE	-	-	-	-	-	-	-
	STD DEV	-	-	-	-	-	-	-

USABILITY (Continued)

Regression analysis not performed.

Reason: R4

Table C-6 Data Collection Summary for Usability (Continued)

SYSTEM A & B		IMPLEMENTATION			
		OP.1	TG.1	CM.1	CM.2
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.83	1	.74	.83
	RANGE	-	-	-	-
	STD DEV	-	-	-	-

MAINTAINABILITY

Regression analysis not performed on the following metrics.

Reasons: R4

Table C-7 Data Collection Summary for Maintenance

SYSTEM A & B		DESIGN			IMPLEMENTATION			
		CS.1	CS.2	MO.1	CS.1	CS.2	SI.2	SD.1
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.99	.8	.53	1	1	1	.25
	RANGE	.75-1.	-	-	-	-	-	.2-.4
	STD DEV	.06	-	-	-	-	-	.05

MAINTAINABILITY (Continued)

Regression analysis performed on following metrics.

Table C-8 Regression Analysis Summary for Maintenance

	DESIGN			IMPLEMENTATION							
SYSTEM B	SI.1	SI.3	MO.2	SI.1	SI.3	SI.4	MO.2	SD.2	SD.3	CO.1	RATING
Average	.72	.68	.78	.69	.68	.77	.68	.74	.82	.78	.32
Range	.4-.99	0-.99	.67-1.	.3-.83	0-.99	.5-.9	.58-.9	.5-1.	.7-.93	.4-.98	.07-.76
Std Dev	.20	.35	.1	.19	.35	.10	.10	.13	.08	.18	.15
Predictor Coefficient	.53	.67	-	.48	.67	.43	.44	.46	.66	.44	
Std Error of Estimate	.27	.28		.15	.28	.17	.16	.15	.31	.24	
Correlation Coefficient	.83	.88		.91	.88	.89	.89	.92	.87	.82	

SYSTEM A	*	**	*	**	**	**	**	**	*	**	RATING
Average		.77		.76	.77	.76	.72	.77	.87	.65	.28
Range		.67-.86		.67-.83	.67-.86	.7-.83	.6-.9	.6-.83	.7-.93	.33-.95	.1-.4
Std Dev		.06		.06	.06	.05	.08	.07	.08	.19	.09
Multiple Regression		✓		✓	✓	✓	✓	✓			Included in set
Predictor Coefficient		.67		.56	.07	0	0	0			
Std Error	.28			Std Error	.24						
Correlation Coefficient	.88			Correlation Coefficient	.89						

* Normalization function rejected.
** Normalization function accepted.

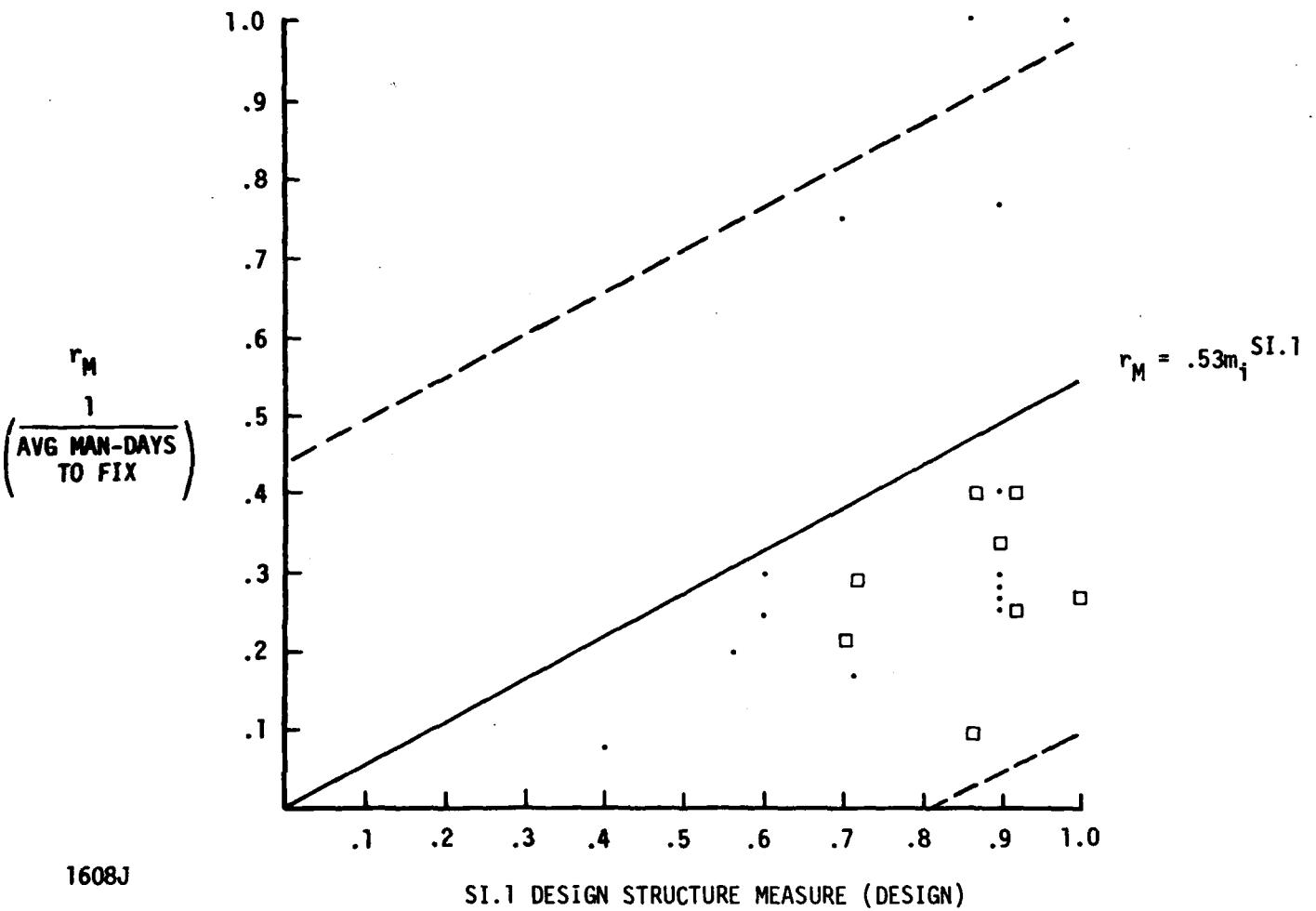
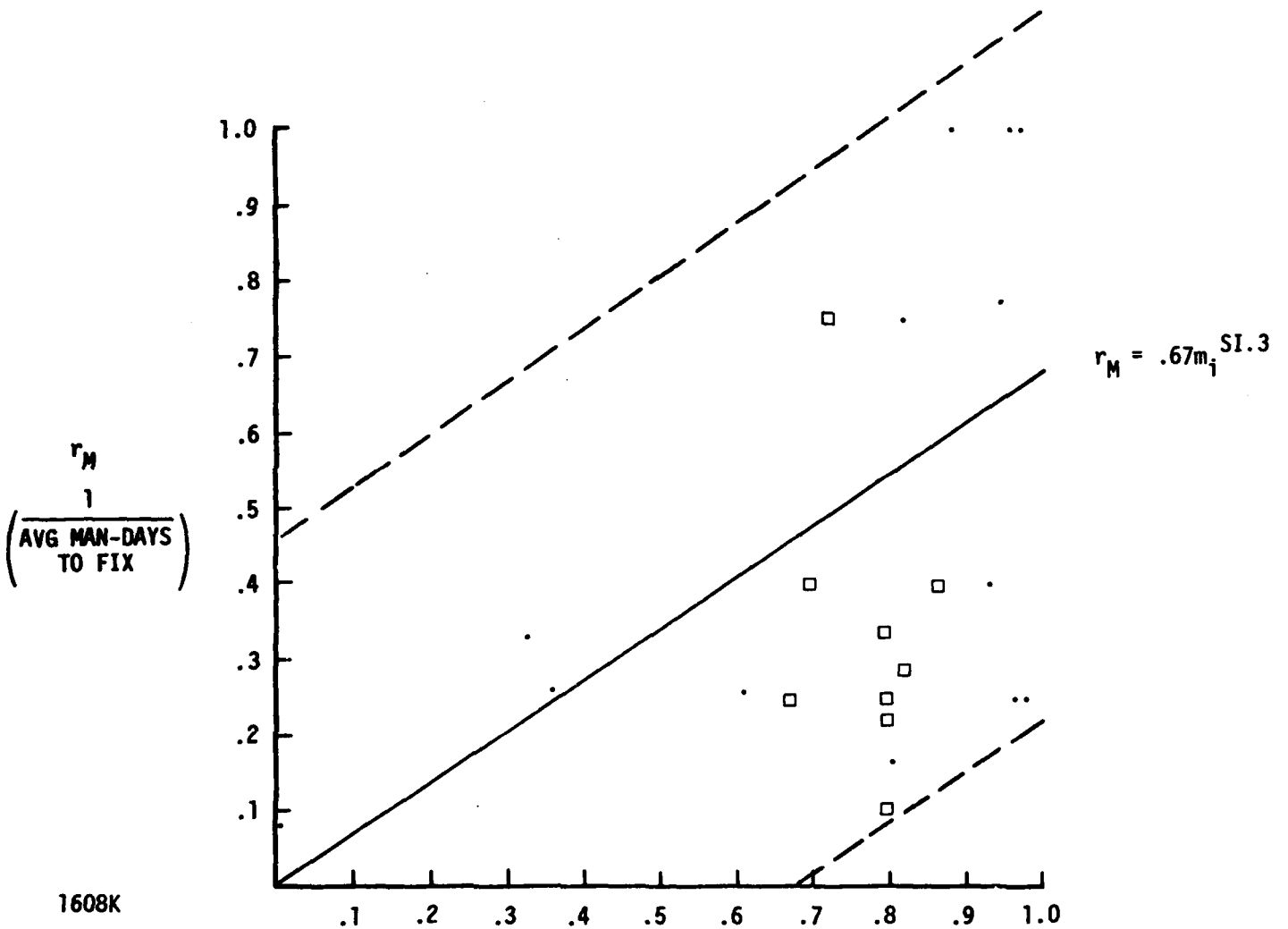


Figure C-10 $SI.1_M$ (Design) Normalization Function



1608K

Figure C-11 $SI.3_M$ (Design and Implementation) Normalization Function

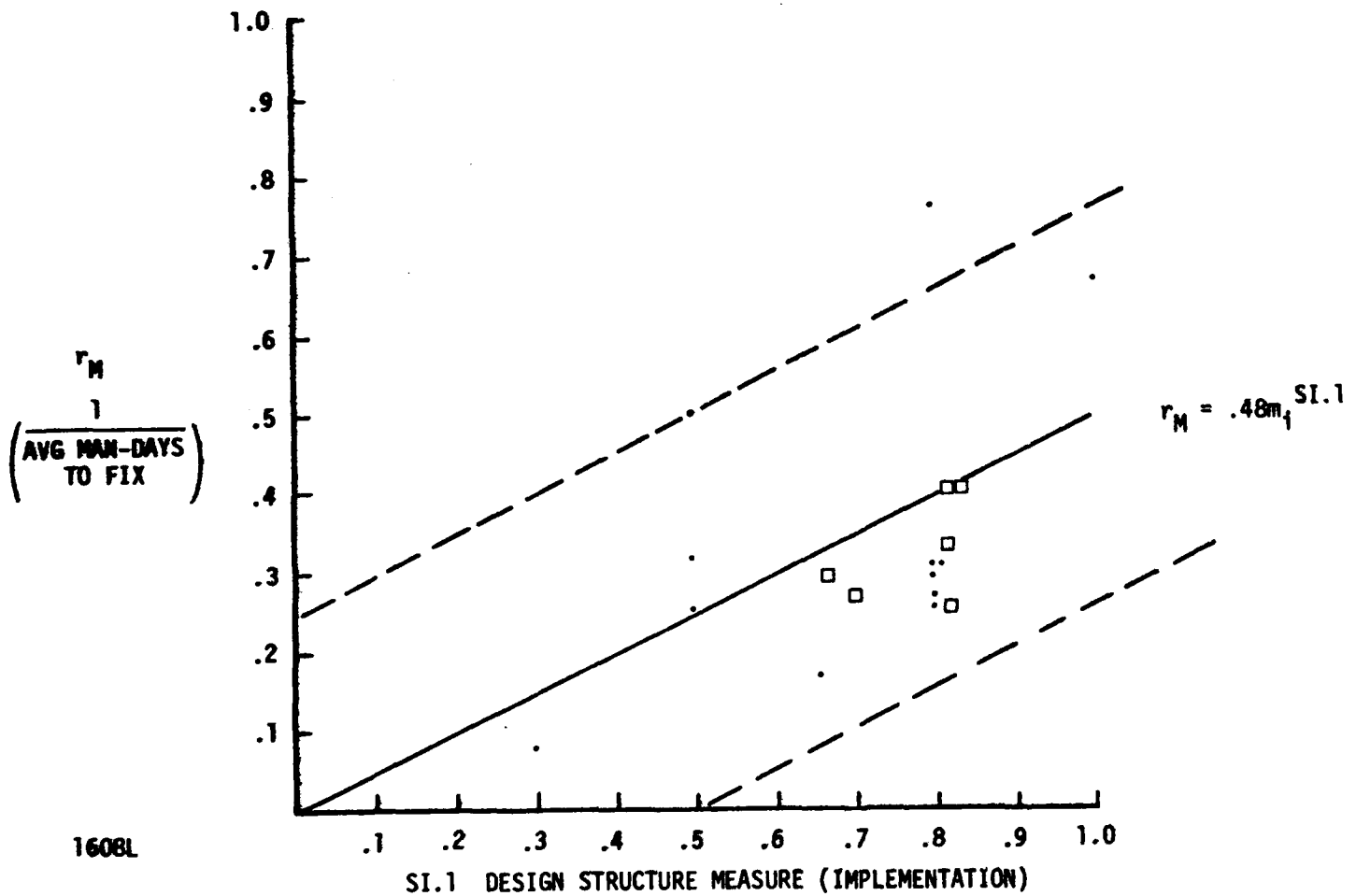


Figure C-12 $SI.1_M$ (Implementation) Normalization Function

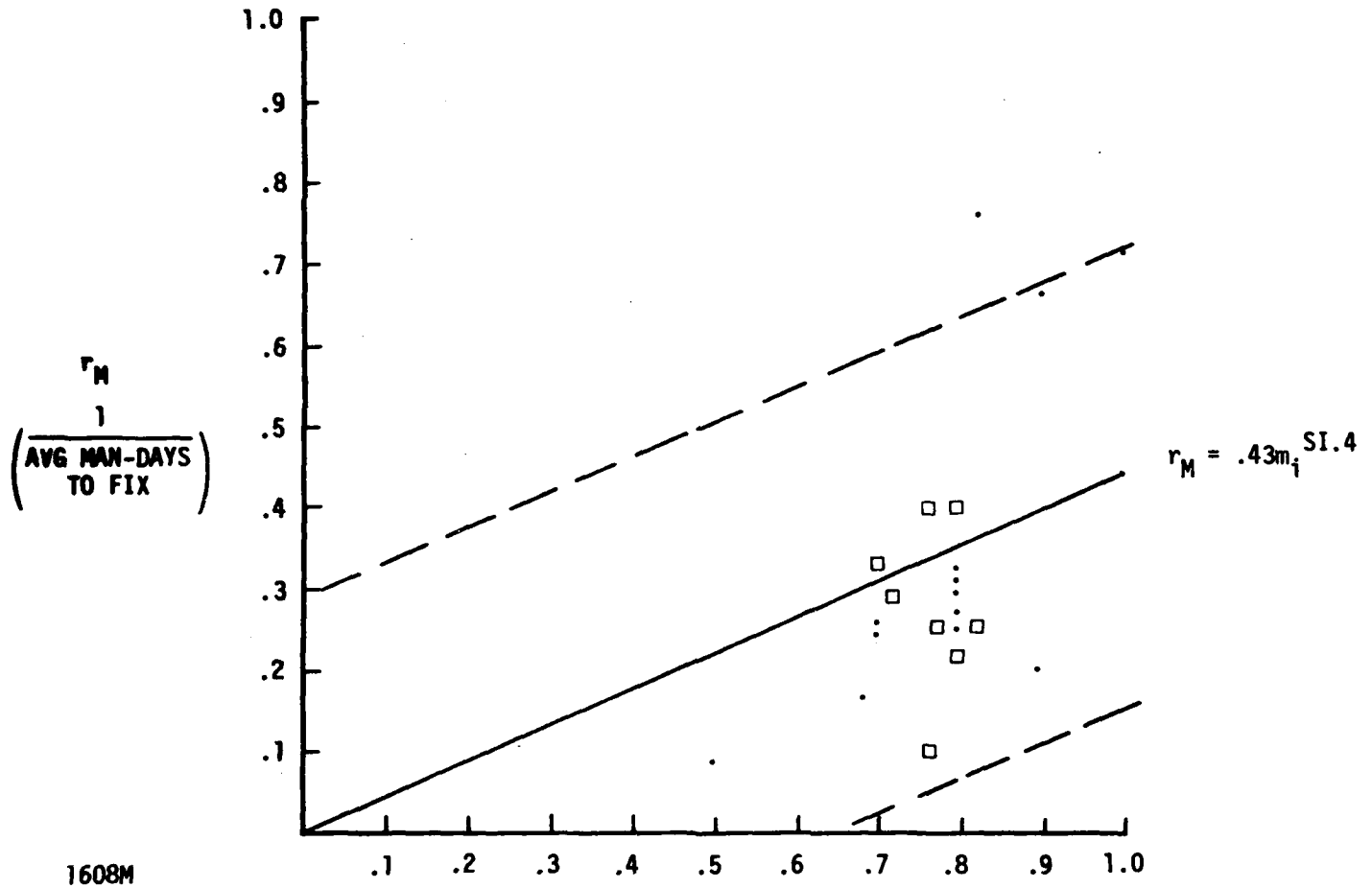


Figure C-13 $SI.4_M$ (Implementation) Normalization Function

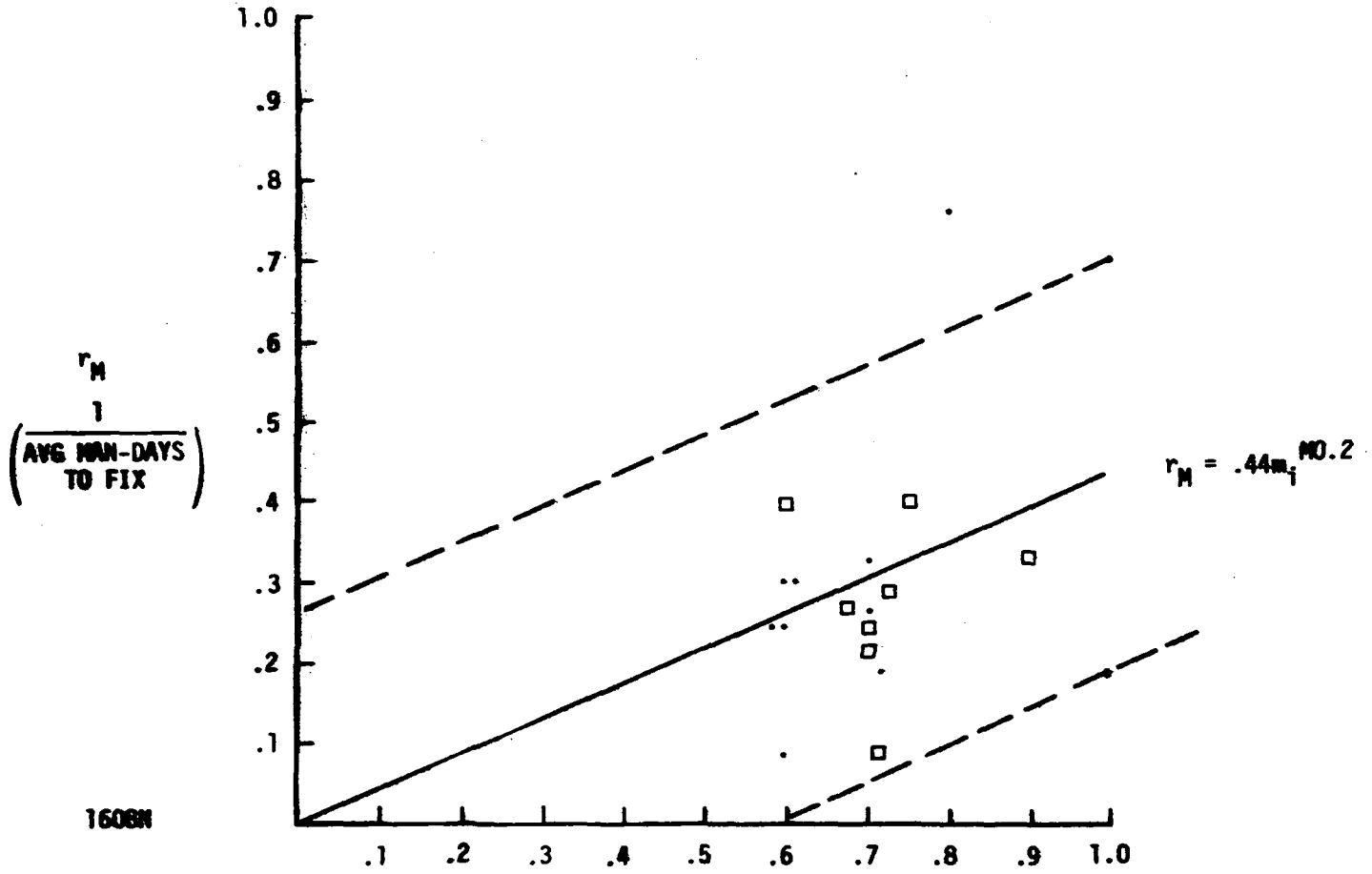


Figure C-14 $MO.2_M$ (Implementation) Normalization Function

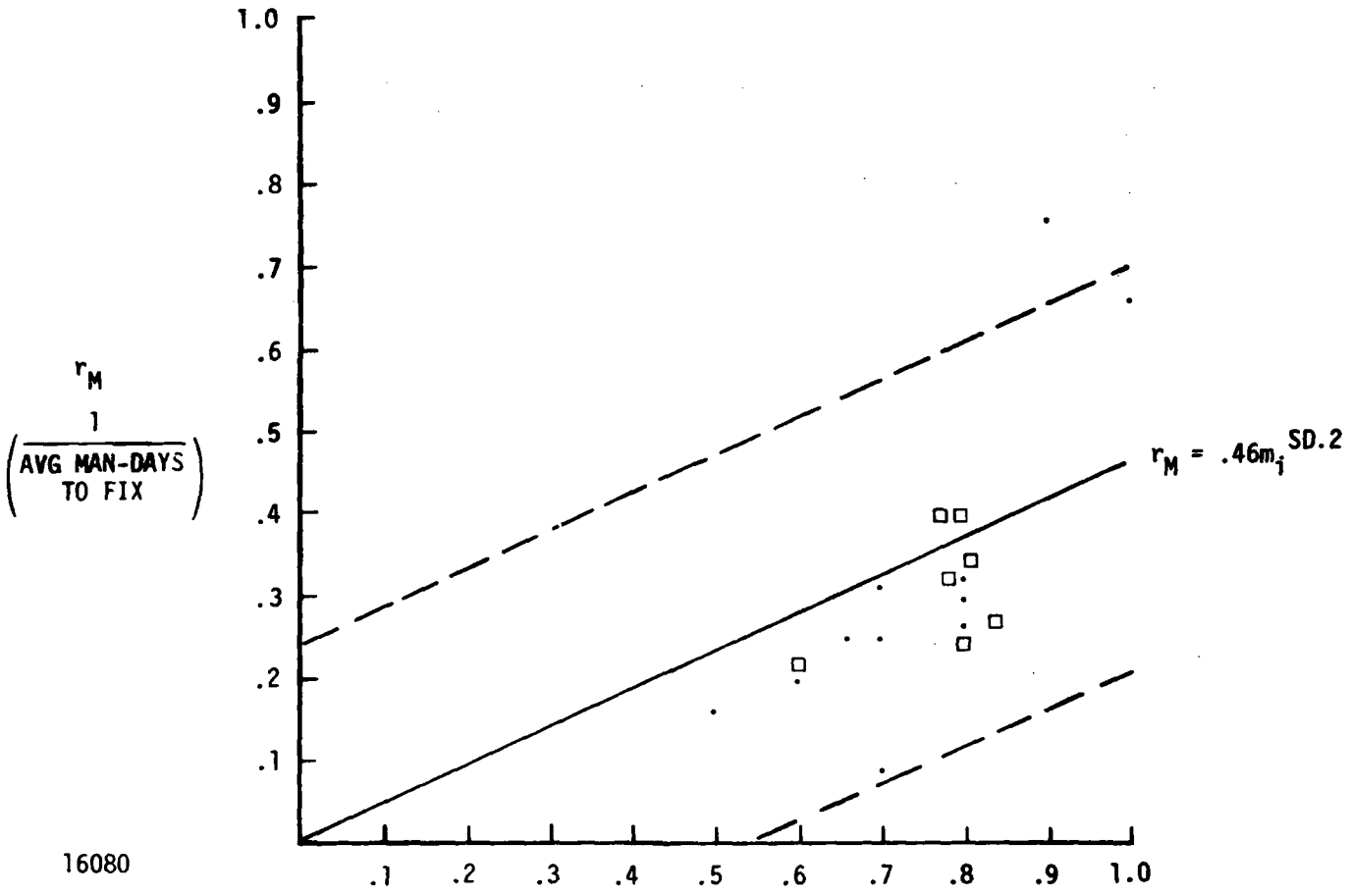
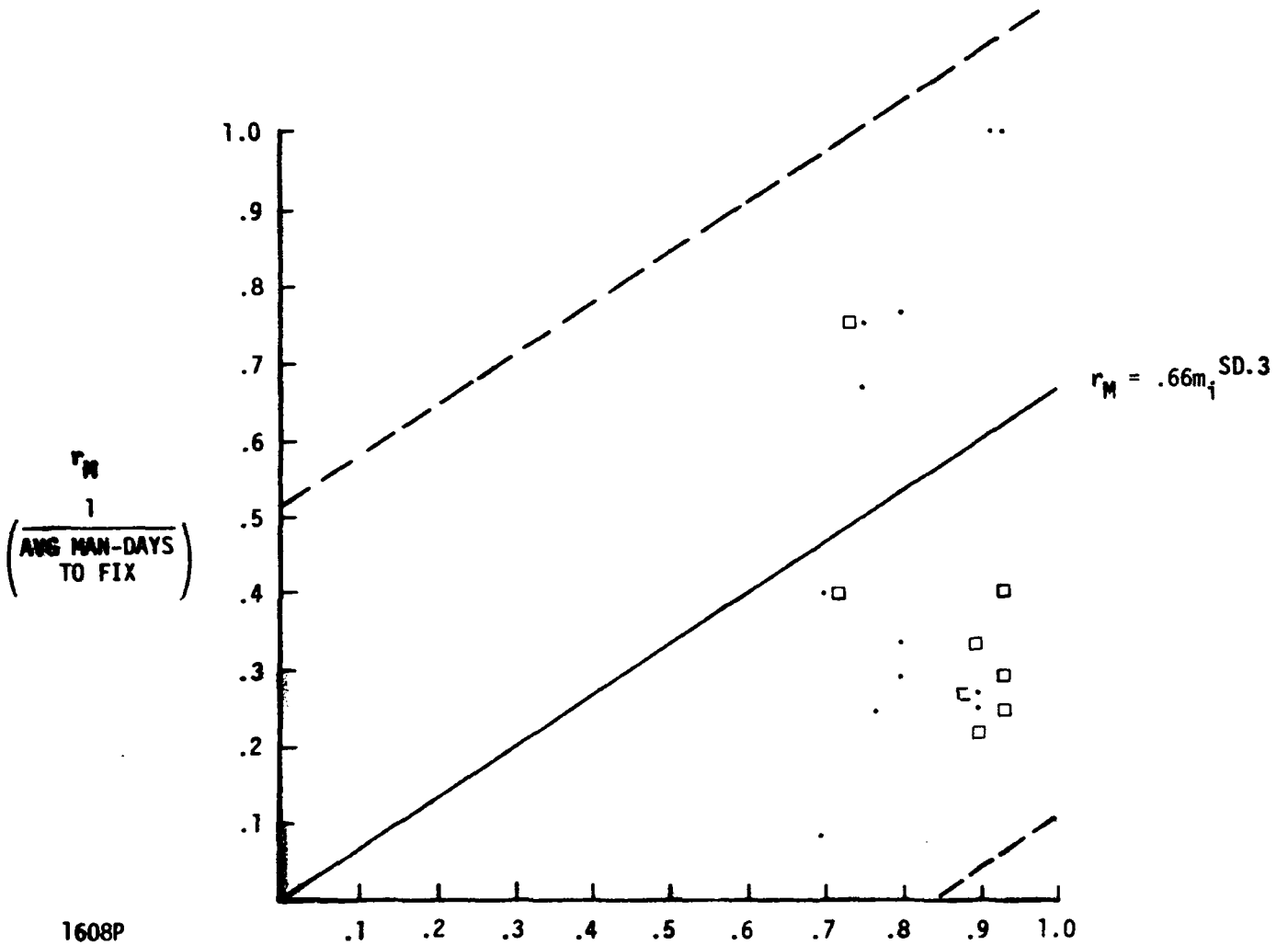


Figure C-15 $SD.2_M$ (Implementation) Normalization Function



SD.3 DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE MEASURE (IMPLEMENTATION)

Figure C-16 $SD.3_M$ (Implementation) Normalization Function

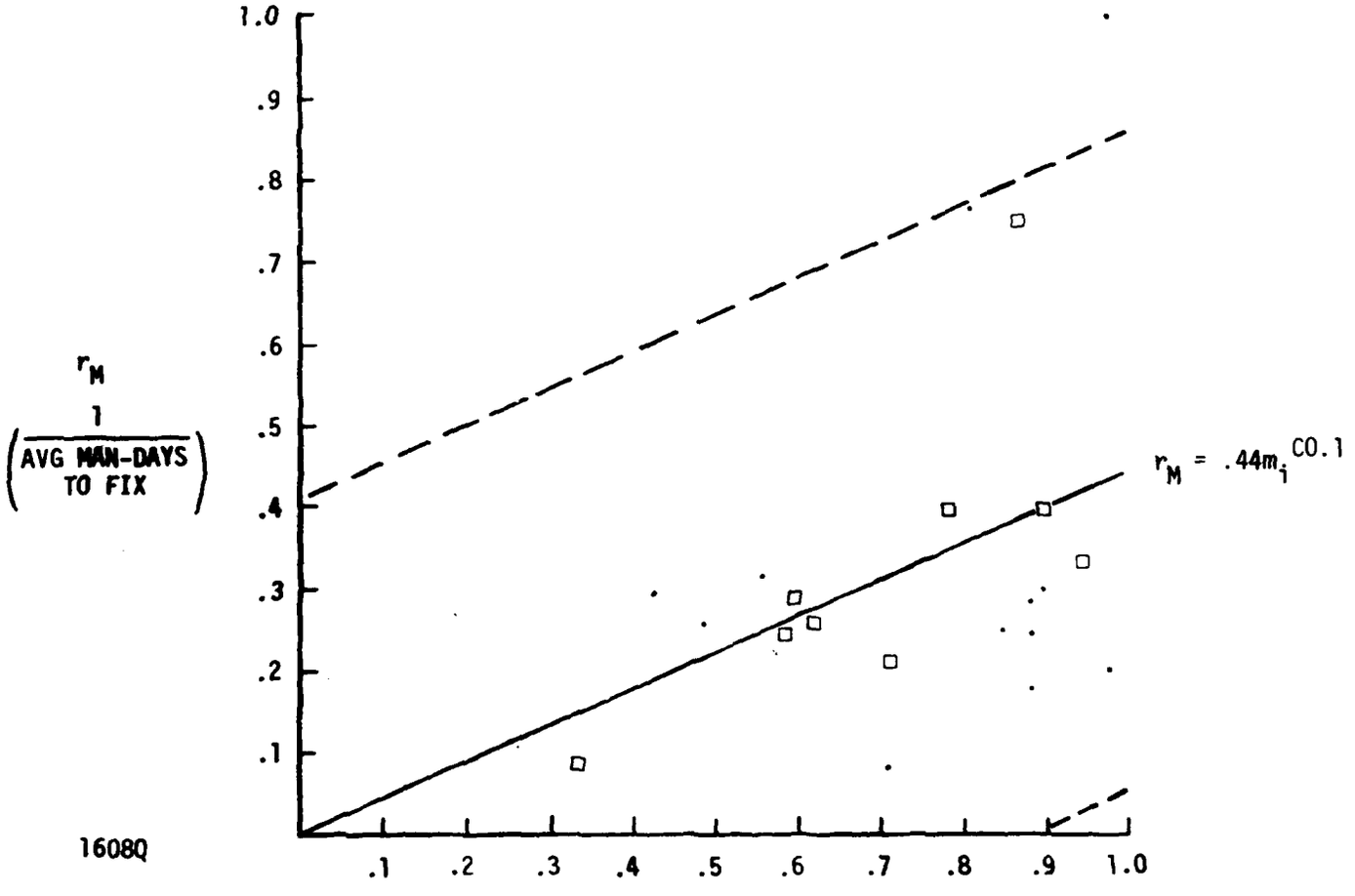


Figure C-17 $CO.1_M$ (Implementation) Normalization Function

FLEXIBILITY

Regression analysis not performed on following metrics.

Reasons: R3, R4

Table C-9 Data Collection Summary for Flexibility

SYSTEM A & B		DESIGN				IMPLEMENTATION			
		MO.1	GE.1	EX.1	EX.2	GE.1	EX.1	EX.2	SO.1
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.53	.32	*	*	.32	*	*	.25
	RANGE	-	.2-.45	-	-	.2-.45	-	-	.2-.3
	STD DEV	-	.13	-	-	.13	-	-	.05

FLEXIBILITY

Regression analysis performed on following metrics.

Table C-10 Regression Analysis Summary for Flexibility

SYSTEM A & B	DESIGN		IMPLEMENTATION				RATING
	MO.2	GE.2	MO.2	GE.2	SD.2	SD.3	
Average	.79	.33	.69	.48	.78	.82	.43
Range	.67-1.	0-1.	.58-.9	.2-.8	.6-.83	.56-.93	.25-.75
Std Dev	.09	.37	.09	.24	.05	.12	.17
Predictor Coefficient	.51	.56	.60	.72	.59	.56	
Std Error of Estimate	.12	.25	.12	.15	.16	.14	
Correlation Coefficient	.96	.75	.96	.93	.95	.96	
Multiple Regression	** ✓	* ✓	** ✓	** ✓	** ✓	** ✓	Included in set
Predictor Coefficient	.51	0	.22	.44	0	.09	
Std Error	.12		Std Error	.11			
Correlation Coefficient	.96		Correlation Coefficient	.98			

*Normalization function rejected.

**Normalization function accepted.

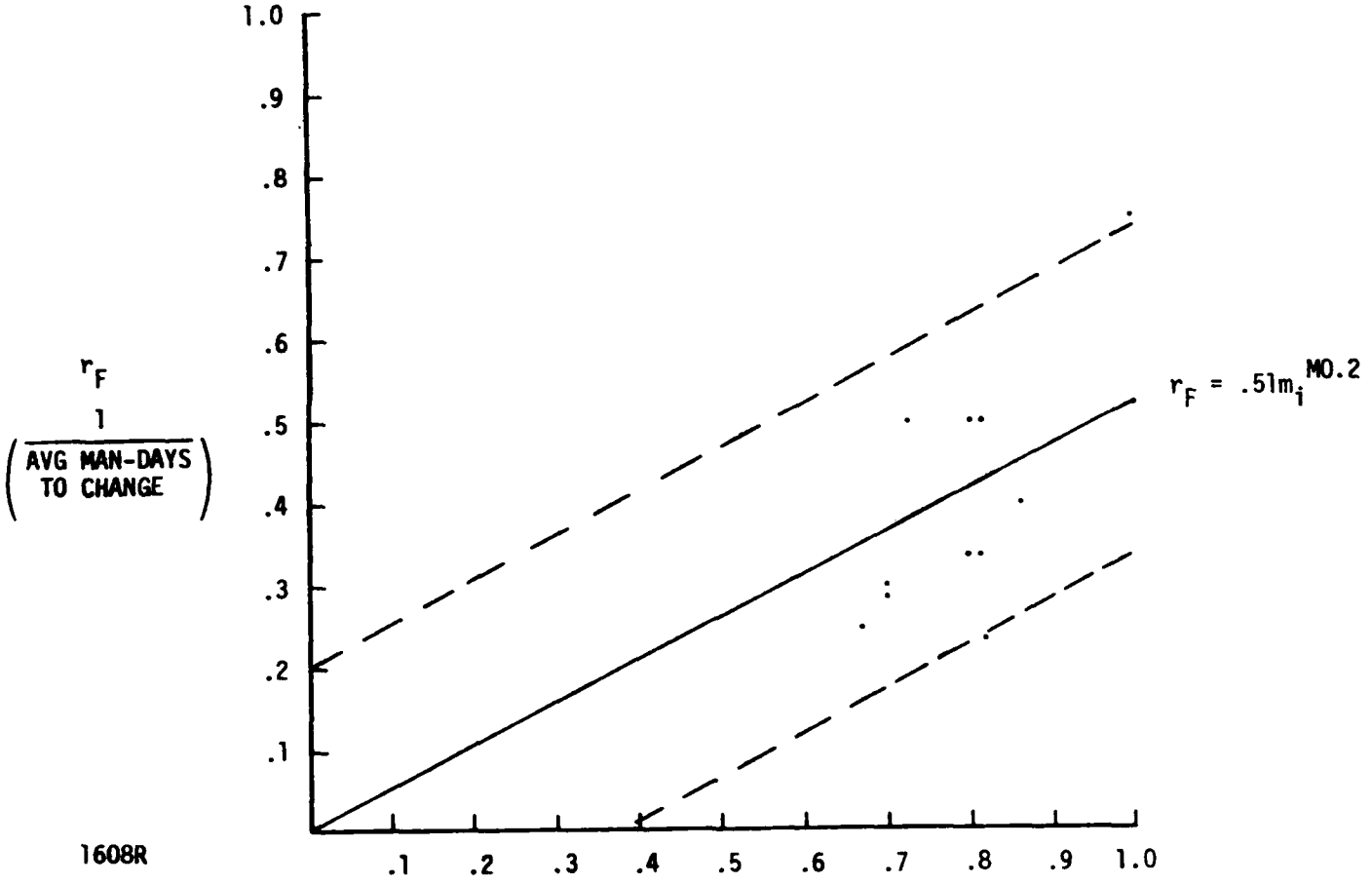
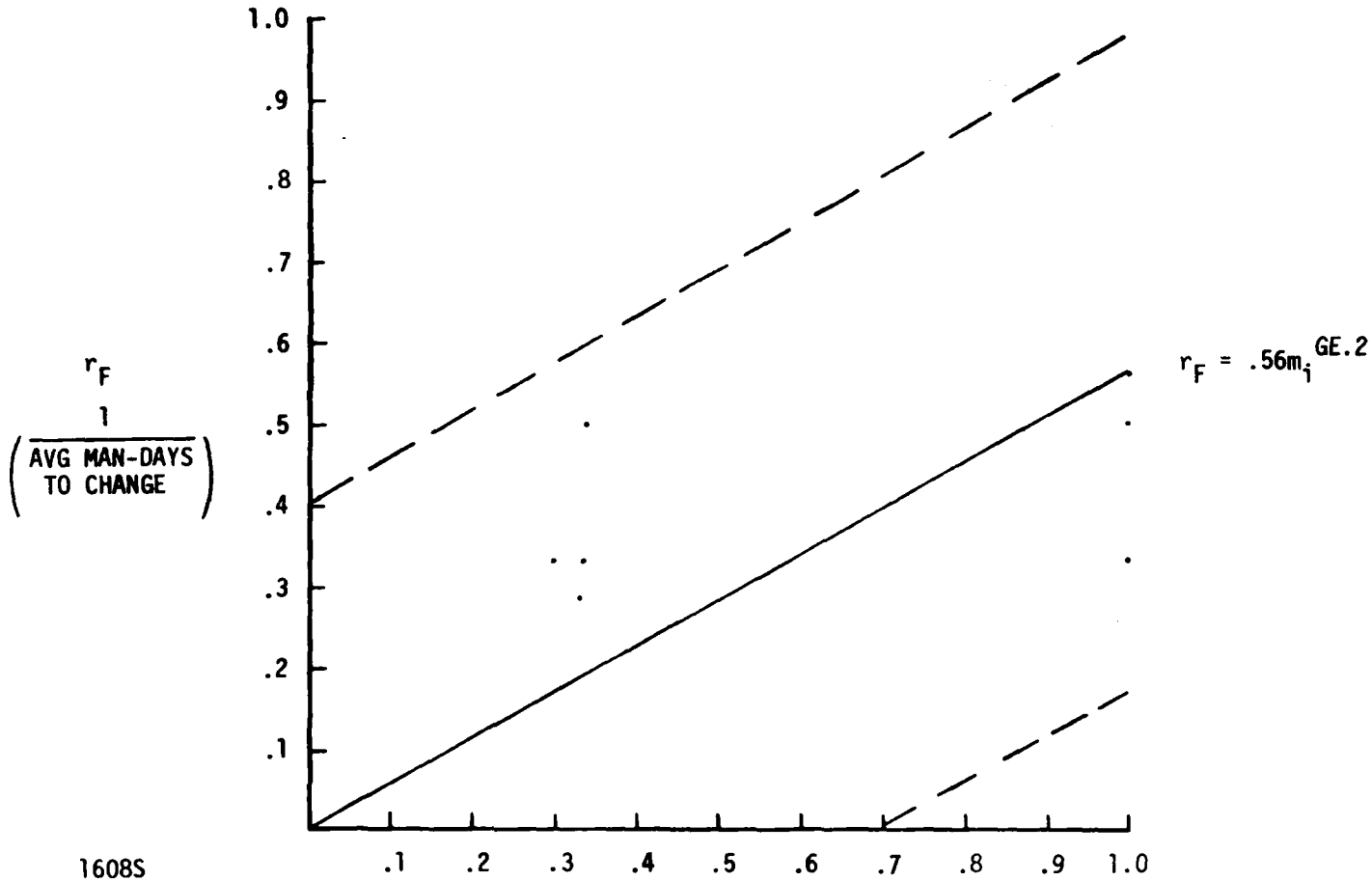


Figure C-18 $MO.2_F$ (Design) Normalization Function



GE.2 IMPLEMENTATION FOR GENERALITY (DESIGN)

Figure C-19 $GE.2_F$ (Design) Normalization Function

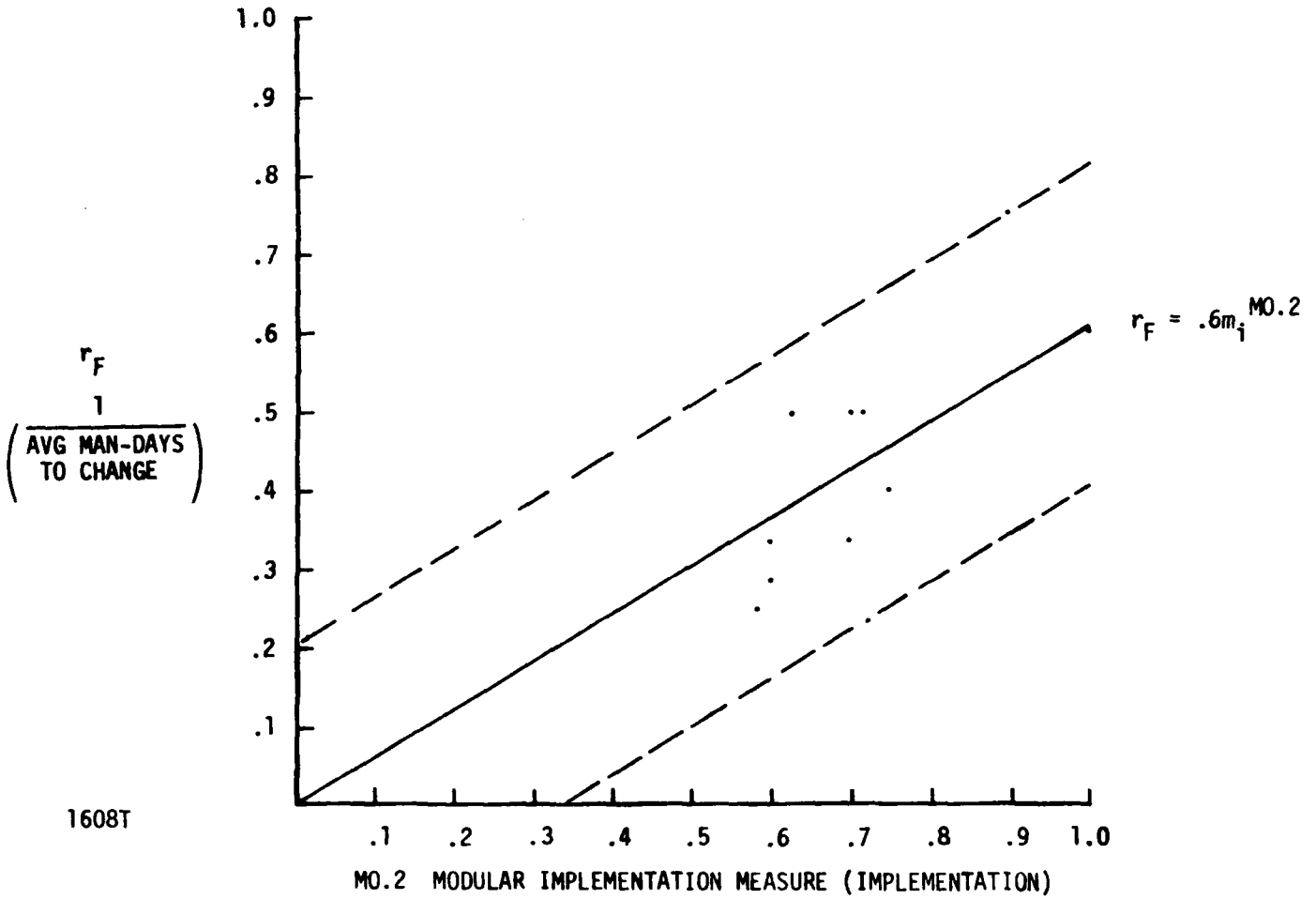


Figure C-20 $MO.2_F$ (Implementation) Normalization Function

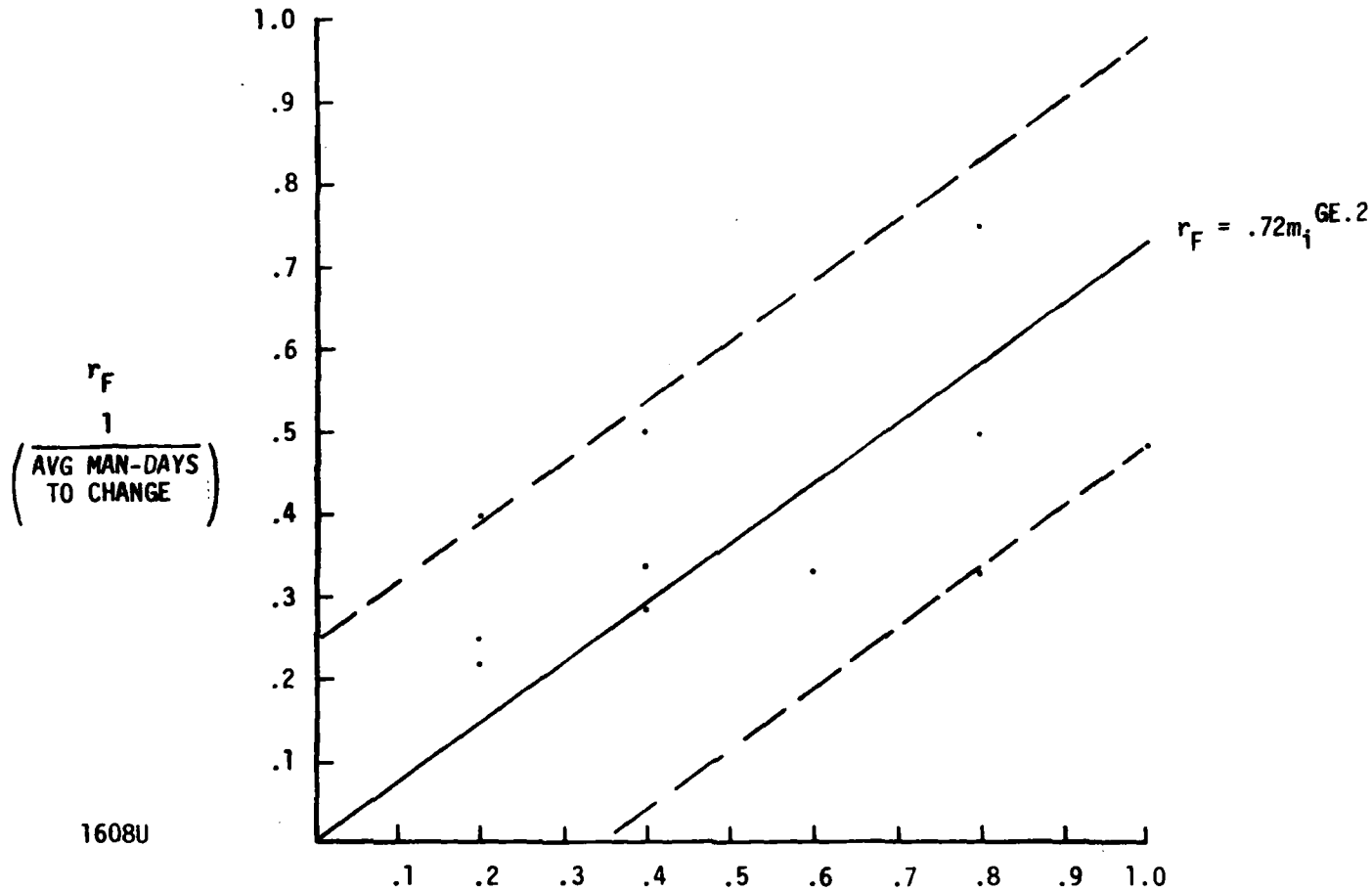


Figure C-21 $GE.2_F$ (Implementation) Normalization Function

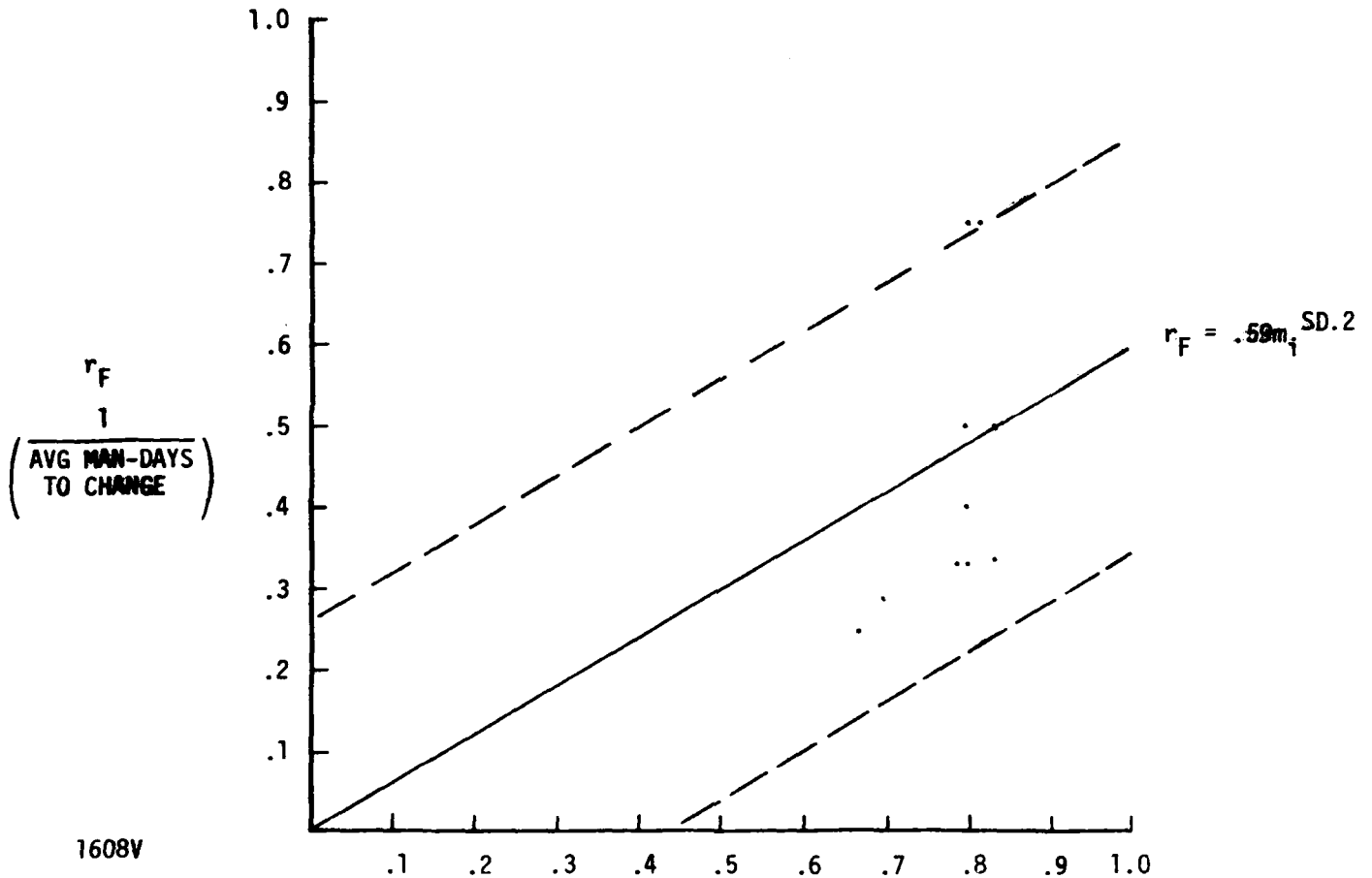


Figure C-22 $SD.2_F$ (Implementation) Normalization Function

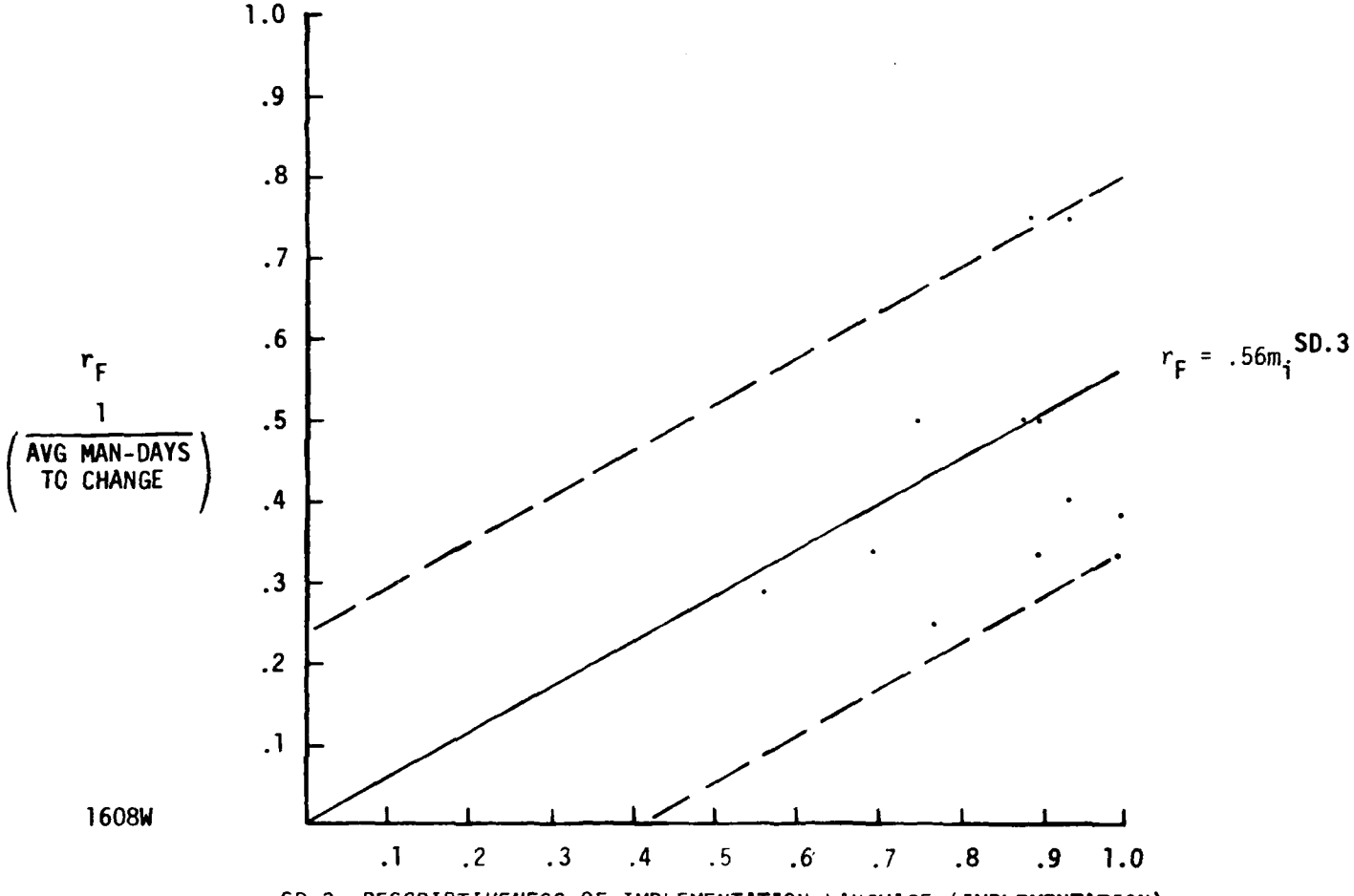


Figure C-23 $SD.3_F$ (Implementation) Normalization Function

TESTABILITY

Regression analysis not performed on the following metrics.

Reasons: R2, R3, R4

Table C-11 Data Collection Summary for Testability

SYSTEM A & B		DESIGN						
		SI.1	SI.3	MO.1	MO.2	IN.1	IN.2	IN.3
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.72	.68	.53	.78	*	*	1
	RANGE	.4-.99	0-.99	-	.67-1.	-	-	-
	STD DEV	.20	.35	-	.1	-	-	-

* Complete information not available.

TESTABILITY (Continued)

Regression analysis not performed on the following metrics.

Table C-11 Data Collection Summary for Testability (Continued)

SYSTEM A & B		IMPLEMENTATION										
		SI.1	SI.2	SI.3	SI.4	MO.2	IN.1	IN.2	IN.3	SD.1	SD.2	SD.3
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.69	1	.68	.77	.68	*	*	1	.25	.74	.82
	RANGE	.3-.83	-	0-.99	.5-.9	.58-9.	-	-	-	.2-.4	.5-1.	.7-.93
	STD DEV	.9	-	.35	.10	.10	-	-	-	.05	.13	.08

* Complete information not available.

REUSABILITY

Regression analysis not performed on the following metrics.

Reasons: R2

Table C-12 Data Collection Summary for Reusability

SYSTEM A & B		DESIGN					
		MO.1	MO.2	GE.1	GE.2	SS.1	MI.1
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.53	.78	.32	.33	.18	0
	RANGE	-	.67-1.	.2-.45	0-1.	.18-.19	-
	STD DEV	-	1.	.13	.37	.01	-

REUSABILITY (Continued)

Regression analysis not performed on the following metrics

Table C-12 Data Collection Summary for Reusability (Continued)

SYSTEM A & B		IMPLEMENTATION							
		MO.2	GE.1	GE.2	SD.1	SD.2	SD.3	SS.1	MI.1
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.68	.32	.48	.69	1	.68	.18	.13
	RANGE	.58-.9	.2-.45	.2-.8	.3-.83	-	0-.99	.18-.19	-
	STD DEV	.10	.13	.24	.19	-	.35	.01	-

PORTABILITY

Regression analysis not performed on the following metrics.

Reasons: R2

Table C-13 Data Collection Summary for Portability

SYSTEM A & B		DESIGN			
		MO.1	MO.2	SS.1	MI.1
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.53	.78	.18	0
	RANGE	-	.67-1.	.18-.19	-
	STD DEV	-	.1	.01	-

PORTABILITY (Continued)

Regression analysis not performed on the following metrics.

Table C-13 Data Collection Summary for Portability (Continued)

SYSTEM A & B		IMPLEMENTATION					
		MO.2	SD.1	SD.2	SD.3	SS.1	MI.1
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.68	.69	1	.68	.18	.13
	RANGE	.58-.9	.3-.83	-	0-.99	.18-.19	-
	STD DEV	.10	.19	-	.35	.01	-

INTEROPERABILITY

Regression analysis not performed on the following metrics.

Reasons: R2

Table C-14 Data Collection Summary for Interoperability

SYSTEM A & B		RQMTS		DESIGN			
		CC.1	DC.1	MO.1	MO.2	CC.1	DC.1
INDIVIDUAL METRIC ANALYSIS	AVERAGE	*	*	.53	.78	*	*
	RANGE	-	-	-	.67-1.	-	-
	STD DEV	-	-	-	.1	-	-

* No Data Available.

INTEROPERABILITY (Continued)

Regression analysis not performed on the following metrics.

Table C-14 Data Collection Summary for Interoperability (Continued)

SYSTEM A & B		IMPLEMENTATION					
		MO.2	SD.1	SD.2	SD.3	CC.1	DC.1
INDIVIDUAL METRIC ANALYSIS	AVERAGE	.68	.69	1	.68	*	*
	RANGE	.58-.9	.3-.83	-	0-.99	-	-
	STD DEV	.10	.19	-	.35	-	-

* No Data Available.

APPENDIX D

METRIC APPLICATION

Each metric will be discussed in this appendix in relation to where it was applied (its source) and how it was collected. Selected examples will be given.

The metrics will be covered in the same order as presented in Table 6.2-1. The legend presented in Table D.1-1 applies to the discussion:

Table D.1-1 Metric Source and Tool Legend

Sources	Code
Software System Requirements Specification	SRS
Standards and Conventions	SC
Documentation Plan	DP
Management Plan	MP
Preliminary Design Specification	PDS
Preliminary Design Review Material	PDR
Detailed Design Specifications	DDS
Critical Design Review Material	CDR
Validation and Acceptance Test Specification	VATS
Users Manual/Operators Manual	UOM
Interface Control Document	ICD
Configuration Management Plan	CMP
Data Base Management Plan	DBMP
Problem Reports	PR
Source Code	CODE
Programmers Notebook	PNK
Training Material	TM

Table D.1-1 Metric Source and Tool Legend (Cont.)

Tools/Techniques Used*	Code
Manual Inspection or Review	MAN
Configuration Management System	CMS
Requirements Trace Routine	RTR
GE/Integrated Software Development System	ISDS
GJSUMRY/ATP - Code Auditor	GJS

* A brief description is provided in paragraph 8.2

Where manual techniques were used but automated tools have been identified that would assist in or are capable of providing the metric data, a code from Table D.1-2 is provided in parentheses. These are briefly described and examples given in paragraph 8.3.

Table D.1-2 Other Data Collection Tools

Automated Tools Available	Code
Requirements Specification Language/Analyzer	PSL
Automated Verification System	AVS
Test Procedure Language	TPL
Program Design Language/Analyzer	PDL
Code Auditor	CA
Execution Analyzer	EA
Consistency Checker	CC
Data Definition Language Processor	DDLP
Data Base Optimizer	DBO

D.1

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
TR. 1 Cross reference relating modules to requirements. $\left(\frac{\#\text{itemized requirements traced}}{\text{total \# requirements}}\right)$			PDS	MAN (PSL)	PDS	MAN (PSL)

The TR.1 measure was determined by a manual analysis of a matrix provided in the preliminary design specification document which identifies which routines satisfy specific software system requirements itemized in the software system requirements specification.

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
CP. 1 COMPLETENESS CHECKLIST:						
(1) Unambiguous references (input, function, output).	SRS	MAN (PSL)	DDS	MAN (PDL)	CODE	MAN (CA)
(2) All data references defined, computed, or obtained from an external source.	SRS	MAN (PSL)	DDS	MAN (PDL)	CODE	MAN (CA)
(3) All defined functions used.	SRS	MAN (PSL)	DDS	MAN (PDL)	CODE	MAN (CA)
(4) All referenced functions defined.	SRS	MAN (PSL)	DDS	MAN (PDL)	CODE	MAN (CA)
(5) All conditions and processing defined for each decision point.	SRS	MAN (PSL)	DDS	ISDS	CODE	MAN (CA)
(6) All defined and referenced calling sequence parameters agree.			DDS	MAN	CODE	MAN
(7) All problem reports resolved.	PR	(CMS)	PR	(CMS)	PR	CMS
(8) Design agrees with requirements.			DDS SRS	MAN (PSL) (PDL)		
(9) Code agrees with design.					DDS CODE	(ISDS) (CA)

Most of the completeness measures (CP.1) were 1, i.e., no deficiencies were detected. This was a function of applying the metrics after delivery. Most of these measures were applied manually during this effort. The use of a formal requirements specification language and formal design language would greatly enhance the automation of the metric data collection for this metric. Automation is definitely required because manual determination of these measures during an on-going project would be a difficult effort. An example of an automated tool used during this effort is the GE/Integrated Software Development System (GE/ISDS) (a description is in paragraph 8.2). GE/ISDS performs analyses on design charts which are prepared via an interactive graphics system and maintained in machine readable form. An example output from an analysis of decision points (CP.1 (5)) is shown in Figure D.2-1.

A tool such as PSL/PSA (CARA) provides the capability to determine such measures as CP.1 (3) and CP.1 (4). Figure D.2-2 provides an excerpt from a configuration management report which was used to determine CP.1 (7).

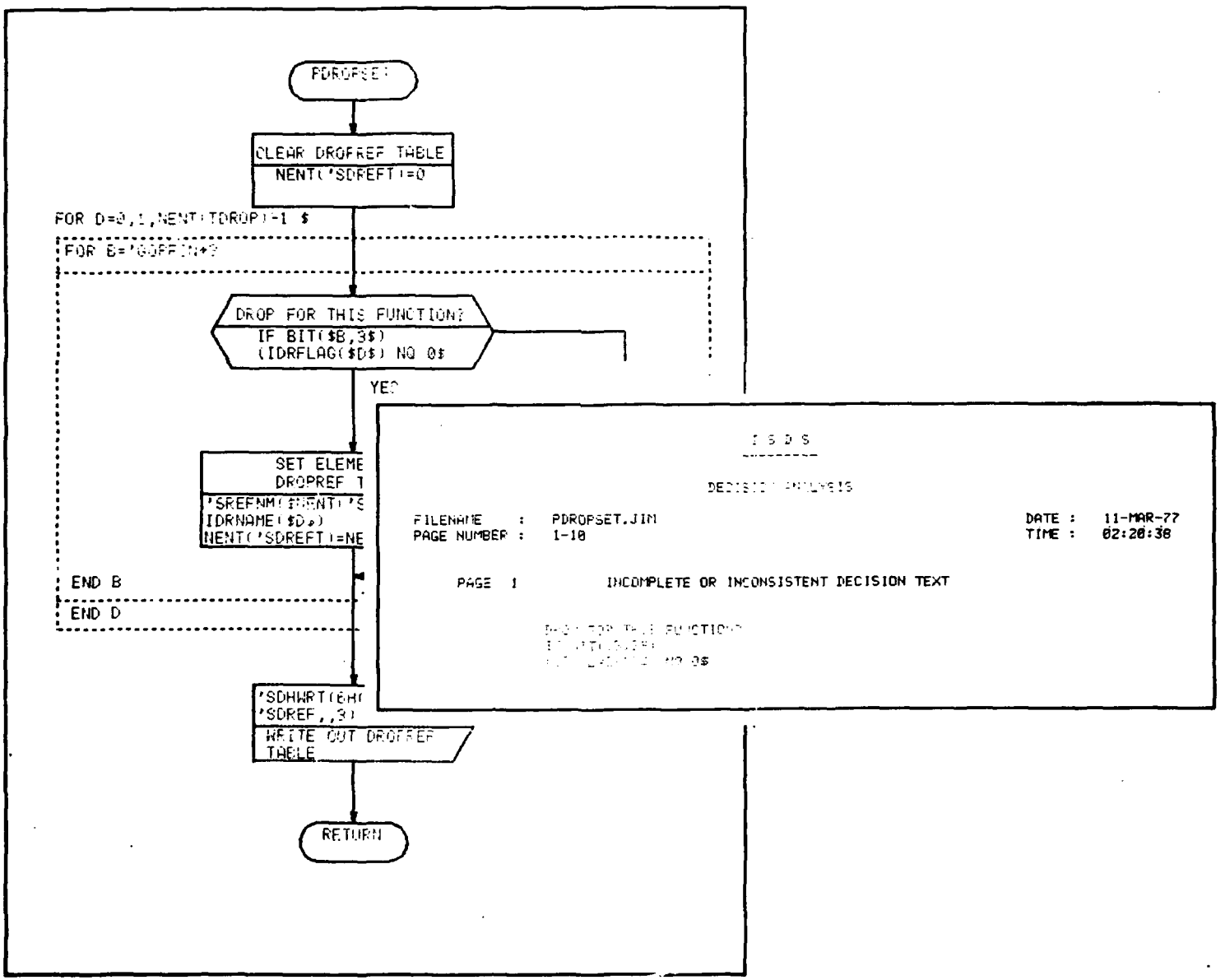


Figure D.2-1 Automated Analysis of Decision Points

BEST AVAILABLE COPY

TASK ID= ANOMA BLOCK 2.2 SPR/MTR DATA NAME SORTED 18NOV76 PAGE 708

SPR CONTROL NUMBER	ISSUE DATE	ELEMENT NAME RESP GU	MOD	PROBLEM DESCRIPTION	STAT	PRI	MTR CONTROL NUMBER	ELEMENT NAME	NEW MOD	DATE CLOSED	ACTION TAKEN	REF CONF	FORM NO.
		DBM										0225	33
												0225	34
0225	39 17DEC74	#GDBU	MD	NEGATIVE MENT CAUSED BOUNDS VIOLATION	ST4	P1	B22M 122	#GDBD	MD	08JAN75	EXPL	0225	39
		DBM											
0225	50 03JAN75	#GDBU	MD	CHECKS SUB-ENTRY HEADER NOT JUSTIFIED PROPERLY	ST4	P1	B22M 171	#GDBUM	ME	30JAN75	AMTR22AV	0225	50
		DBM											
S45A	6085 21FEB75	#GDBU	MD	11 DIGITS MAX OUTPUT FLT PT	ST3	P3	M45A 416	#GDBD	MD	07APR75	STAT	S45A	6085
		DBM											
S45A	6099 13MAR75	#GDBU	MD	DBM CONFLICTS COMPOOL DEFINED CONVERSIONS	ST4		B22M 343	#GDBU	ME	24MAR75	EXPL	S45A	6099
		DBM										S45A	6125
S45A	6100 13MAR75	#GDBU	MD	A TYPE ITEMS DO NOT DISPLAY PROPERLY	ST4	P2	M45A 404	#GDBGE	MB	04APR75	AMTB22BE	S45A	6100
		DBM											
0225	302 26MAR75	#GDBUS	ME	LISTELAL CAND ALGORITHM ERROR	ST4		M45A 409	#GDBDS	MF	04APR75	AMTB22BE	0225	302
		DBM											
0225	6 29OCT74	#GDBUV	MB	ERROR OCCURS IN EVALUATING TEST RESULTS FROM #GDBGM	ST4		B22M 36	#GDBDV	MC	31OCT74	AMTB22AK	0225	6
		DBM											

Figure D.2-2 Status of Problem Reports

BEST AVAILABLE COPY

D.3

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
CS. 1 PROCEDURE CONSISTENCY MEASURE						
(1) Standard design representation $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	ISDS		
(2) Calling sequence conventions $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS ICD	ISDS	CODE	MAN (CA)
(3) Input/output conventions $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	ISDS	CODE	MAN (CA)
(4) Error handling conventions $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN (ISDS)	CODE	MAN (CA)

Enhancements to many of the typical currently utilized code audit routines could display for easier inspection or enforce conventions relating to the CS.1 elements identified. During design, GE/ISDS is an example of automated analysis performed on design charts relating to these conventions. Enhancements to a tool such as ISDS could provide complete coverage of this metric. Examples are shown in Figure D.3-1.

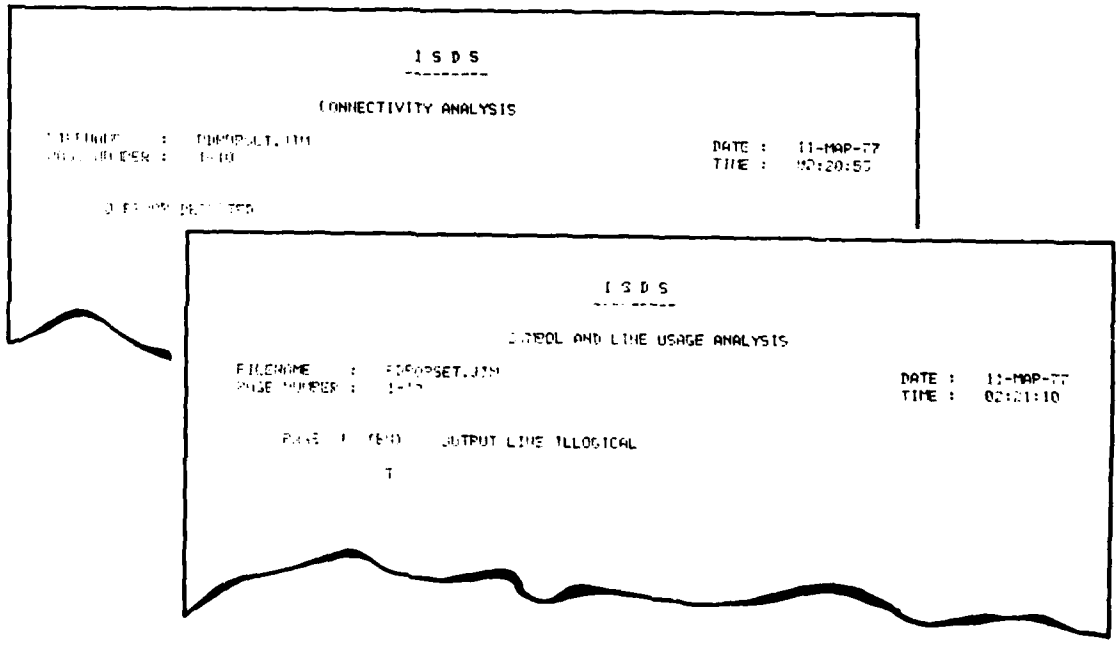


Figure D.3-1 Automated Consistency Checks During Design

D.4

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
CS. 2 DATA CONSISTENCY MEASURE						
(1) Standard data usage representation $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN (ISDS)		
(2) Naming conventions $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN (ISDS) (DDLP)	CODE	MAN (DDLP) (CA)
(3) Unit consistency $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(4) Consistent global definitions $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS DBMP	MAN	CODE	MAN (DDLP) (CA)
(5) Data type consistency $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN (CA)

The measures dealing with data consistency (CS.2) pose a very difficult manual data collection task. The use of an automated tool is necessary both from a cost to collect and an accuracy standpoint. The use of a formal Program Design Language would enhance the ability to automate the collection of this data. For this effort, the Data Base Management Plan, the section of the Detailed Design Specifications which identified the internal variables to be used in a routine, and the organization and substantial commenting of the code aided the manual collection of this metric data. A current enhancement to the design charts produced during contractual efforts to include data representations will allow automated checking of CS.2(1), as shown in Figure D.4-1.

D.4 (continued)

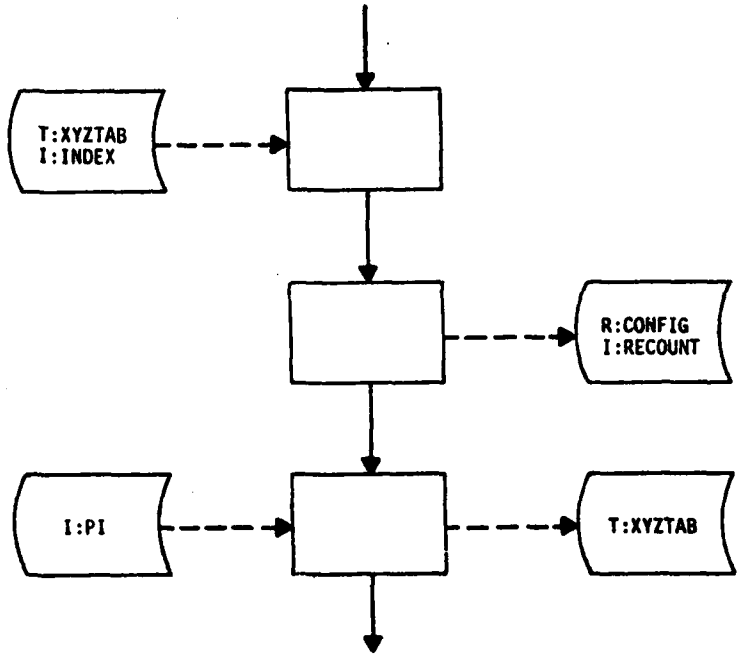


Figure D.4-1 Data Representation on Flowcharts

The type of problem that should be identified by CS.2(4), consistent global definitions, is illustrated by this example:

```
Routine A
     COMMON/VAR/SUM, DEV, ROOT
     COMMON/MATRIX/X(15), Y(25)

Routine B.
     COMMON/VAR/TOTAL, ROOT, DEV
     COMMON/MATRIX/X(25), Z(15)
```

The Data Base Management Plan and adherence to the plan in the design documents is the key to prevention of these types of errors.

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
AC. 1 ACCURACY CHECKLIST:						
(1) Error analysis performed and budgeted to module.	SRS	MAN				
(2) A definitive statement of requirement for accuracy of inputs, outputs, processing, and constants.	SRS	MAN				
(3) Sufficiency of math library.			DDS	MAN (EA)		
(4) Sufficiency of numerical methods.			DDS	MAN	CODE	MAN
(5) Execution outputs within tolerances.					CODE	MAN (EA)

The accuracy checklist measures (AC.1) require a high level analyst familiar with the mathematical requirements of the system to manually inspect and analyze the documents and described methods. Some assistance can be derived from analysis of execution or simulation results.

An example presented in [VANTD74] illustrates t alysis required for AC.1(4). Three implementations of the equation;

$$y = Ax^3 + Bx^2 + Cx + D$$

are provided. They are listed in order of increasing efficiency and accuracy.

```
y = A*X**3 + B*X**2 + C*X + D
y = A*X*X*X + B*X*X + C*X + D
y = D + X*(C + X*(B+A*X))
```

D.6

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
ET. 1 ERROR TOLERANCE CONTROL CHECKLIST:						
(1) Any concurrent processing centrally controlled.			PDS DDS	MAN	CODE	MAN
(2) Errors should be fixable and processing continued.			PDS DDS	MAN	CODE	MAN
(3) When an error condition is detected, it should be passed up to calling routine.			PDS DDS	MAN	CODE	MAN
ET. 2 RECOVERY FROM IMPROPER INPUT DATA CHECKLIST:						
(1) A definitive statement of requirement for error tolerance of input data.	SRS	MAN				
(2) Range of values (reasonableness) for items specified and checked.			DDS	MAN	CODE	MAN
(3) Conflicting requests and illegal combinations identified and checked.			DDS	MAN	CODE	MAN
(4) All input is checked before processing begins.			DDS	MAN	CODE	MAN
(5) Determination that an data is available prior to processing.			DDS	MAN	CODE	MAN
ET. 3 RECOVERY FROM COMPUTATIONAL FAILURES CHECKLIST:						
(1) A definitive statement of requirement for recovery from computational failures.	SRS	MAN				
(2) Loop and multiple transfer index parameters range tested before use.			DDS	MAN	CODE	MAN
(3) Subscript checking.			DDS	MAN	CODE	MAN
(4) Critical output parameters reasonableness checked during processing.			DDS	MAN	CODE	MAN
ET. 4 RECOVERY FROM HARDWARE FAULTS CHECKLIST:						
(1) A definitive statement of requirement for recovery from hardware faults.	SRS	MAN				
(2) Recovery from hardware faults (e.g., arithmetic faults, power failure, clock).			DDS	MAN	CODE	MAN
ET. 5 RECOVERY FROM DEVICE ERRORS CHECKLIST:						
(1) Definitive statement of requirement for recovery from device errors.	SRS	MAN				
(2) Recovery from device errors.			DDS	MAN	CODE	MAN

D.6 (continued)

All of the measures associated with error tolerance (ET.1 through ET.5) were manually collected. The major source of information was the description of the inputs, processing, and limitations in the detailed design specification. Often, an itemized limitation would contain a phrase such as; "all function cards must be reinput under an error condition," or, "the permissible values are," or, "the acceptable range," etc. These phrases identify limitations on the processing. In order for the system to be error tolerant, it should "gracefully" handle violations of these limitations. Most of the measurements are oriented toward the identification of failures to provide typical error tolerant capabilities.

The software system requirement specification was also a valuable source. In our environment, an Operational Hardware/Software Specification evolves during the development phase from the System Requirements Specification. The operational constraints imposed by both hardware and software are described. The error tolerant handling of the constraints must be considered. The manual inspection of the code is considerably easier if there is a good detail design document and accurate design charts. These documents aid in identifying the key portions of the code to inspect. Basically input processing code should be examined to insure necessary checks occur. Loop indices and subscript checking, if necessary, is easily checked. The common example given is:

```
IVAR = 0
DO 100  I = 1,N
IVAR = IVAR+1
  .
  .
  .
100 CONTINUE
```

If $N \leq 0$, the wrong value of IVAR is realized. Instead a check should be made, such as:

D.6 (continued)

```
IVAR = 0
IF (N.LE.Ø)
   DO 100  I = 1,N
   IVAR = IVAR+1
     .
     .
     .
   100 CONTINUE
END IF
```

D.7

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
SI. 1 DESIGN STRUCTURE MEASURE:						
(1) Design organized in top down fashion.			PDS DDS	MAN (ISDS)	CODE	MAN (CA)
(2) No duplicate functions.						
(3) Independence of module $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(4) Module processing not dependent on prior processing. $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(5) Each module description includes input, output processing, limitations. $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN (PDL)		
(6) Each module has single entrance, single exit. $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	ISDS	CODE	GJS
(7) No global data.			DDS	MAN	CODE	MAN (CA)

The design documentation consists of a preliminary design specification (PDS) and a detailed design specification (DDS) (see Appendix B for a description). The detailed design specification is usually organized with an overview and subsystem descriptions and then descriptions at the program level. The preliminary design specification and detailed design overview were the major sources for determining the design structure metric (SI.1), during the design phase. A hierarchy chart and brief descriptions of the subsystems and planned modules were used to determine if top down design techniques had been followed. Several violations were found for SI.1(3), (4), and (6). Most of these violations were identified by examining the section of the detailed design specifications which describe the calling sequence. Several modules had multiple entrances (GE/ISDS flags this attribute from the design charts). Several modules were dependent on prior processing. The most common indication of this was "the following data blocks must be set prior to execution of this module" phrase found in the calling sequence description. This is a characteristic of a COMPOOL system but is also evident in any environment where there is considerable (SI.1.(7)) global data.

D.8

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
SI. 2 USE OF STRUCTURE LANGUAGE OR PREPROCESSOR					CODE	MAN

Refer to paragraph 7.3.

D.9

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
SI. 3 COMPLEXITY MEASURE (by module, see para. 6.2.2.6)			DDS	ISDS	CODE	GJS

GE/ISDS was utilized to calculate a complexity measure (SI.3) from a special form of a design chart. This measure could be calculated from machine readable design charts, a PDL, or from code, however, these enhancements have not been implemented in ISDS yet. Instead, data gathered by a code auditing routine, GJSUMRY, was utilized to calculate a modified version of Halstead's E measure [HALSM73,HALSM72,LOVET76a]. This measure is based on the number of operators and operands in a program.

An example of the automated calculation of the ISDS generated complexity measure is in Figure D.9-1.

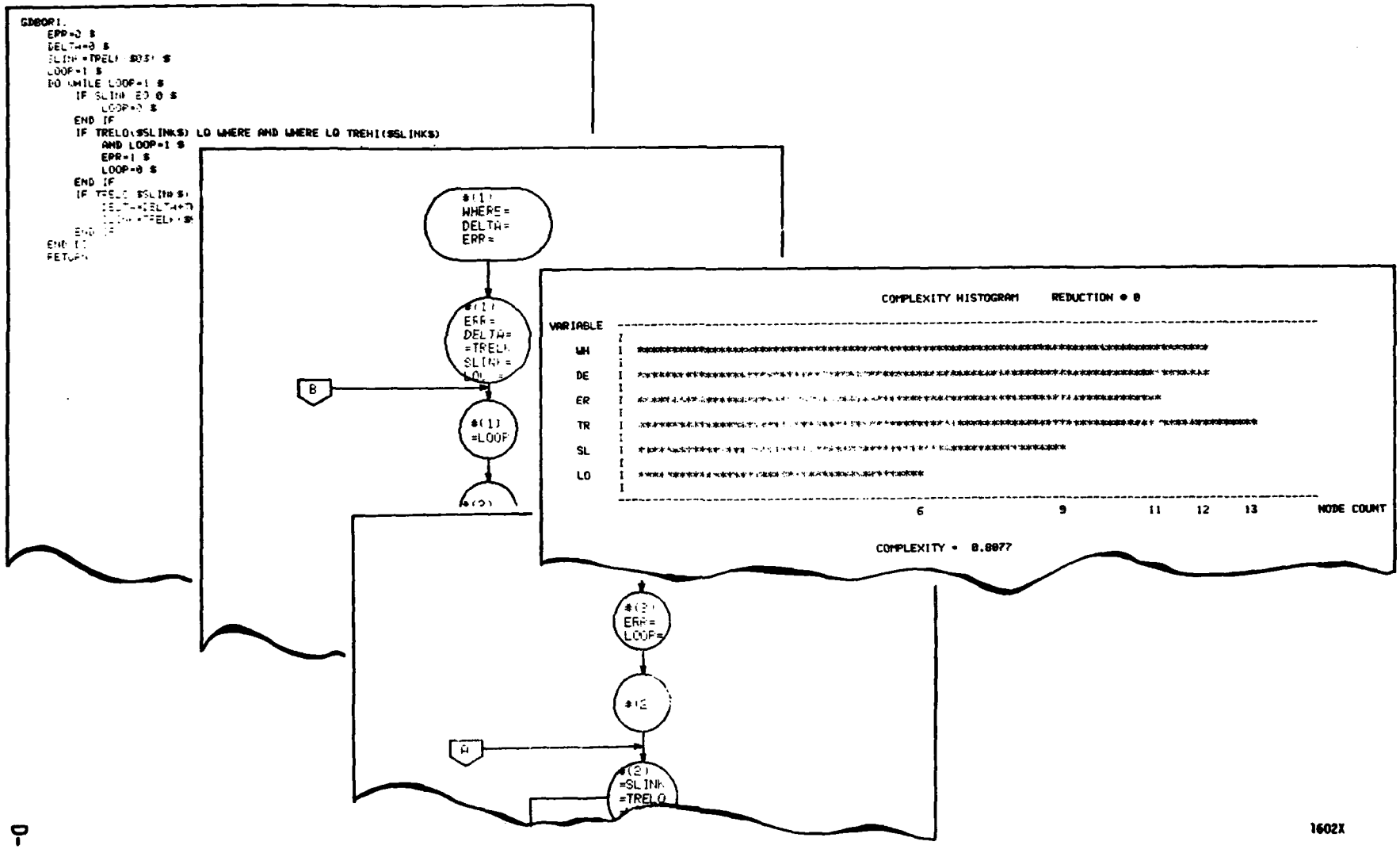


Figure D.9-1 GE/ISDS Complexity Measure

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
SI. 4 MEASURE OF CODING SIMPLICITY (by module)						
(1) Module flow top to bottom.					CODE	MAN
(2) Negative Boolean or complicated compound Boolean expressions used. $\left(1 - \frac{\text{\# of above}}{\text{\# executable statements}}\right)$					CODE	MAN
(3) Jumps in and out of loops $\left(\frac{\text{\# single entry/single exit loops}}{\text{total \# loops}}\right)$					CODE	GJS
(4) Loop index modified $\left(1 - \frac{\text{\# loop indices modified}}{\text{total \# loops}}\right)$					CODE	MAN
(5) Module is not self-modifying.					CODE	MAN
(6) All arguments passed to a module are parametric.					CODE	MAN
(7) Number of statement labels. $\left(1 - \frac{\text{\# labels}}{\text{\# executable statements}}\right)$					CODE	MAN (CA)
(8) Unique names for variables.					CODE	MAN
(9) Single use of variables.					CODE	MAN
(10) No mixed mode expressions.					CODE	MAN
(11) Nesting level $\left(\frac{1}{\text{max nesting level}}\right)$					CODE	MAN (CA)
(12) Number of branches $\left(1 - \frac{\text{\# branches}}{\text{\# executable statements}}\right)$					CODE	GJS
(13) Number of GOTOs $\left(1 - \frac{\text{\# GOTO statements}}{\text{\# executable statements}}\right)$					CODE	GJS
(14) No extraneous code exists.					CODE	MAN
(15) Variable mix in a module $\left(\frac{\text{\# internal variables}}{\text{total \# variables}}\right)$					CODE	MAN (CA)
(16) Variable density $\left(1 - \frac{\text{\# variables}}{\text{\# executable statements}}\right)$					CODE	MAN (CA)

The code simplicity measurements (SI.4) represent a mix of manual inspection of the source code and a report of statistics about the source code by a code audit routine. Several other elements could have been collected automatically (as indicated) with some minor enhancements to the code audit routine. Most of these manual measurements, once a routine is established, can be done quite efficiently. Approximately 1-2 man-hours were spent per routine taking the manual measurements relating to the source code.

D.10 (continued)

Figure D.10-1 displays examples of output from two code audit routines used. The annotations describe some of the data collected.

An example of SI.4(2), complicated BOOLEAN expressions, shown here;

```
IF ≠GRLERR EQ V(NOREC) $
   BEGIN                                          ≠≠BEGIN 3 ≠≠
   IF ≠GDEREQ NQ V(DEL) $
      BEGIN                                       ≠≠BEGIN 4 ≠≠
      IF(≠SDACCS EQ V(RANFIX) AND ≠GDERRF GR ≠SDMAXN OR
      ≠GDTTDT($0$) NQ V(VBLK) AND(≠GDERRF NQ 0 OR ≠GDERRF
      NQ 0)) $
         BEGIN                                    ≠≠BEGIN 5 ≠≠
```

illustrates the complexity it adds to the program. Their use may also complicate the logic as shown by [KERNB74] with this example;

```
IF (.NOT. FLAG) THEN A=B ELSE X=Y;
```

which is simplified as

```
IF (FLAG) THEN X=Y ELSE A=B;.
```

The standards and conventions we established for JOVIAL restrict the use of the switch construct because it constitutes self-modification. An example of the complexity self-modifying code adds is given by [YOURE75] with this COBOL example (ALTER in COBOL is similar to SWITCH in JOVIAL).

```
      ALTER SWITCH1 TO PRCCEED TO SWITCH2
        .
        .
        .
SWITCH1
      GO TO INITIALIZATION-ROUTINE
        .
        .
        .
SWITCH2
      ALTER SWITCH3 TO PROCEED TO SWITCH4
        .
        .
        .
```

BEST AVAILABLE COPY

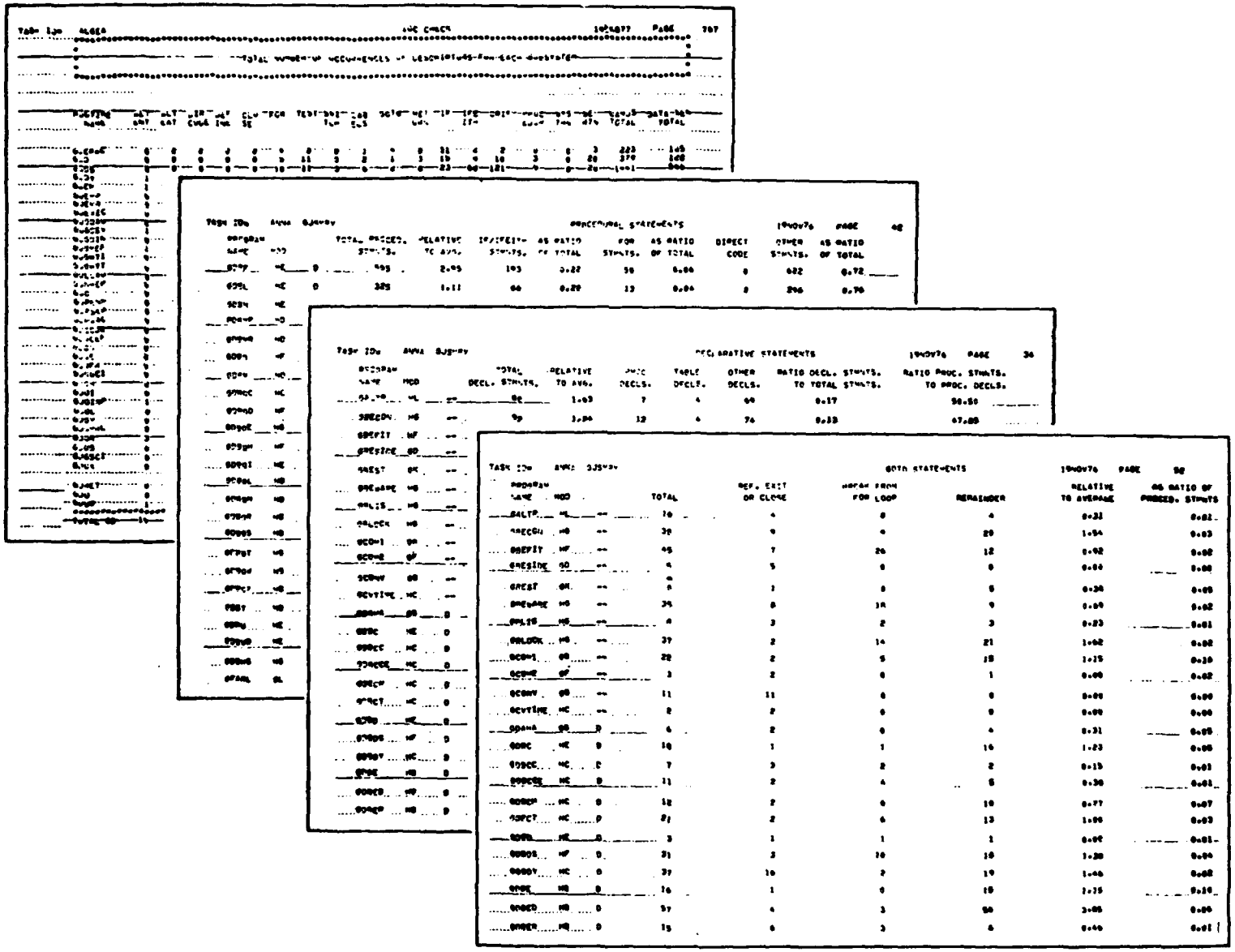


Figure D.10-1 Source Code Profiles by Code Audit Routines

D.10 (continued)

[BOEHM73a] provides justification for SI.4(7) with this example:

> The label, 5, is unnecessary.
>
> ```
>    DO 10  I = 1,N
>  5 X(I) = A(I)+B(I)
> 10 CONTINUE
> ```

SI.4(8), (9), (10) posed the biggest data collection problem. Certainly, examples such as:

```
A(1)    stores craft altitude
A(2)    stores craft velocity
A(3)    stores date of run

        and

TOTAL
SUM     } all same quantities used in different modules
CUM
```

given in [BOEHM73a], and

```
DIMENSION A(20), B(30)
  :
  :
DO 10  I = 1,50
10 A(I) = J
```

by [YOUR75) are important to catch. The greatest assistance in applying these metrics is well commented declarative statements which identify the attributes of the variables. An example from a module inspected during this study is shown here:

```
**************************************************************************
*                                                                        *
********************************** ITEMS *********************************
*                                                                        *
**************************************************************************
)##
     ITEM   SEC       0 $     ## SAVED SYS ERR COUNT  ##
     ITEM   SAVPLK     0 8 $   ## SAVE BLOCK NAME  ##
     ITEM   SAVFILE    I 48 U $ ## SAVE FILE NUMBER  ##
     ITEM   SAVREC     I 48 U $ ## SAVE RECORD NUMBER  ##
     ITEM   OPERANS    0 8 $   ## OPERATOR RESPONSE  ##
     ITEM   SZERO      0 $     ## MUST BE ZERO  ##
     ITEM   BLEN       0 $     ## BLOCK LENGTH  ##
```

D.11

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
NO. 1 STABILITY MEASURE $\left(\frac{\text{expected \# modules changed}}{\text{total \# modules}}\right)$			PDS DDS	MAN (ISDS)		

The stability measure is based on Myer's [MYERG75] stability model. Each module is categorized according to specific criteria as to its module strength and module coupling. A matrix is built based on values assigned the various categorizations and the module interactions within the system. Taking into account the various paths by which a change to one module may effect another module, a second order matrix is calculated. From this second order matrix, a prediction of the number of modules that can be expected to be effected if any module is changed is calculated. Since this is a system level metric, validation was impossible. The value derived for System B, for example was very high compared to the examples presented by Myers. This is very logical because in our environment, most data is global to the system (COMPOOL) and accessed by many routines. Thus any change to the data or the way the data is manipulated by a routine potentially can effect a large number of other modules in the system. The significance of the measure is even if other modules are not effected, considerably more effort must be expended testing to insure they are not in this type of an environment.

D.12

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
MO. 2 MODULAR IMPLEMENTATION MEASURE						
(1) Hierarchical structure $\left(1 - \frac{\text{\# violations of hierarchy}}{\text{total \# modules}}\right)$			PDS DDS	MAN (ISDS)	CODE	MAN (CA)
(2) All modules do not exceed standard module size (100) $\left(1 - \frac{\text{\# modules} > 100}{\text{total \# modules}}\right)$					CODE	GJS
(3) All modules represent one function $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(4) Controlling parameters defined by calling module $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(5) Input data controlled by calling module $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(6) Output data provided to calling module $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(7) Control returned to calling module $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(8) Modules do not share temporary storage			DDS	MAN	CODE	MAN (CA)

Each of the modular implementation measures except for MO.2(2), were collected manually during this effort. The detail design specifications identified module interactions by a called/calling matrix and had a specific section which described the calling sequence and parameters for each module. An evaluation of the parameters and their effect on the processing led to the above measures.

In MO.2(1), any interactions between modules which are not successors or predecessors on the same branch are considered violations. In Figure D.12-1, the measure is 3 violations/8 modules = .375. The violations are the interaction between modules 1 - 2, between 1 - 5 and 2 - 5, and between 5 - 6.

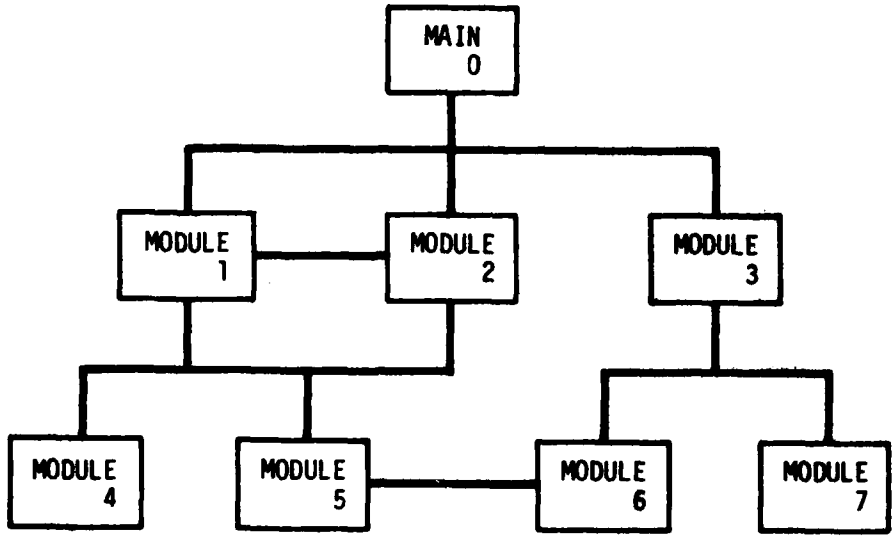


Figure D.12-1 Hierarchical Structure Measure Example

D.13

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
GE. 1 EXTENT TO WHICH MODULE IS REFERENCED BY OTHER MODULES $\left(\frac{\text{\# common modules}}{\text{total \# modules}}\right)$			DDS	MAN (ISOS) (PDL)	CODE	MAN (CA)

This measurement was also easily determined by the same source mentioned in D.12 and confirmation made by inspection of the code.

The implication of this measure is that modules which interface with a number of other modules in their current application will probably be easier to interface or use in another application.

D.14

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
GE. 2 IMPLEMENTATION FOR GENERALITY CHECKLIST						
(1) Input, processing, output functions are not mixed in a single module. $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$			DDS	MAN (ISDS)	CODE	MAN (CA)
(2) Application and machine-dependent functions are not mixed in a single module. $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$					CODE	MAN (CA)
(3) Processing not data volume limited. $\left(1-\frac{\#\text{ modules limited}}{\text{total }\#\text{ modules}}\right)$			DDS	MAN	CODE	MAN (EA)
(4) Processing not data value limited. $\left(1-\frac{\#\text{ modules limited}}{\text{total }\#\text{ modules}}\right)$			DDS	MAN	CODE	MAN (EA)
(5) All constants should be defined once. $\left(1-\frac{\#\text{ modules violate rule}}{\text{total }\#\text{ modules}}\right)$					CODE	MAN (CA)

Since GE.2 is a system level metric, no validation was possible. However, the characteristics identified by these measures, we feel, are quite important to the generality of the software produced. (1) and (2) were determined by a simple identification in the design documents and code whether the module contained any machine dependent, input, and/or output code. The more modules involved in machine dependent functions or input or output, the more effort will be required to use any one module in another environment or application.

GE.2 (3) and (4) correspond to several error tolerant elements where the intent of the measures was to determine how well the system handles exception cases. These measures are related to the fact that there are exception cases. The more limitations on a module or a system the less generally usable it is. This includes the case where the algorithm or function and its implementation are restrictive or the case where poor programming practices have restricted the general use of the program. An example of the later situation is if the limit of a loop in a module which represents the maximum number of inputs to that module is "hard-coded", it is more difficult to change the module correctly to handle more input.

D.14 (continued)

```
      DO 10  I = 1,25
         SUM = SUM + X(I)
   10 CONTINUE
```

A better implementation from a generality viewpoint would be to make the limit flexible with respect to the number of inputs.

```
      DU 10  I = 1,N
         SUM = SUM + X(I)
   10 CONTINUE
```

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
EX. 1 DATA STORAGE EXPANSION MEASURE:						
(1) Logical processing independent of storage specification/requirements (by module) $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(2) Percent of memory capacity uncommitted $\left(\frac{\text{amount of memory uncommitted}}{\text{total amount of available memory}}\right)$					CODE	(EA)

EX.1(2) requires execution of the code under typical loading conditions. Most job execution reports provided by operating system software provides the data to determine this measure.

Examples, from System B and [YOURE75], of using parameters to insure the processing is independent of the storage specification follow:

```
FOR  N = NENT(TRELOC) $
   BEGIN
      .
      .
      .
   END

*THIS ALC PROGRAM REQUIRES THREE TABLES: THE SIZE OF
*EACH TABLE IS A FUNCTION OF THE PARAMETER
*"SIZE".TO CHANGE THE SIZE OF THE TABLES,
*MERELY REDEFINE "SIZE".
*
SIZE EQU 40
TABLE1  DS  CL(SIZE)
TABLE2  DS  CL(2*SIZE)
TABLE3  DS  CL(SIZE+5)
```

D.16

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
EX. 2 EXTENSIBILITY MEASURE:						
(1) Accuracy, convergence, timing attributes which control processing are parametric. $\left(1 - \frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(2) Modules table driven. $\left(1 - \frac{\#\text{ modules not table driven}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN
(3) Percent of speed capacity uncommitted. $\left(\frac{\text{amount of cycle time uncommitted}}{\text{total processing time}}\right)$					CODE	(EA)

The discussion in paragraph D.15 pertains to EX.2(3) also. EX.2(1) and (2) require an analysis of the design strategy expressed in the design document. The following example is described in [YOURE75].

```
c
c     THIS IS A SUBROUTINE THAT WILL SEARCH THROUGH ANY
c     SINGLE-DIMENSIONED ARRAY TO FIND A SPECIFIED
c     ARGUEMENT.  THE ARGUEMENTS TO THE SUBROUTINE
c     ARE AS FOLLOWS:
c          TABLE   THE NAME OF THE ARRAY TO BE SEARCHED
c          FIRST   THE LOWER DIMENSION OF THE ARRAY
c          LAST    THE UPPER DIMENSION OF THE ARRAY
c          ARG     THE QUANTITY BEING SEARCHED FOR
c          FLAG    INDICATES WHETHER OR NOT SEARCH WAS SUCCESSFUL

      SUBROUTINE SEARCH (TABLE, FIRST, LAST, ARG, FLAG)
      DIMENSION TABLE (FIRST, LAST)
        .
        .
        .
        .
```

D.17

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
IN. 1 MODULE TESTING MEASURE (by module)						
(1) Path coverage. $\left(\frac{\text{\# paths to be tested}}{\text{total \# paths}}\right)$			DDS PNK	MAN ISDS	CODE	(AVS) (TPL)
(2) All input parameters boundary tested. $\left(\frac{\text{\# parameters to be boundary tested}}{\text{total \# parameters}}\right)$			DDS PNK VATS	MAN	CODE	(AVS) (TPL)
IN. 2 INTEGRATION TESTING MEASURE						
(1) Module interfaces tested. $\left(\frac{\text{\# to be tested}}{\text{total \# interfaces}}\right)$			ICD VATS	MAN	CODE	(AVS) (TPL)
(2) Performance requirements (timing & storage) coverage. $\left(\frac{\text{\# requirements to be tested}}{\text{total \# perf requirements}}\right)$			SRS VATS	MAN RTR	CODE	(AVS) (TPL)
IN. 3 SYSTEM TESTING MEASURE						
(1) Module coverage (for all test scenarios) $\left(\frac{\text{\# modules to be tested}}{\text{total \# of modules}}\right)$			SRS VATS	MAN	CODE	(AVS) (TPL)
(2) Identification of test inputs and outputs in summary form.			DDS VATS	MAN	CODE	MAN

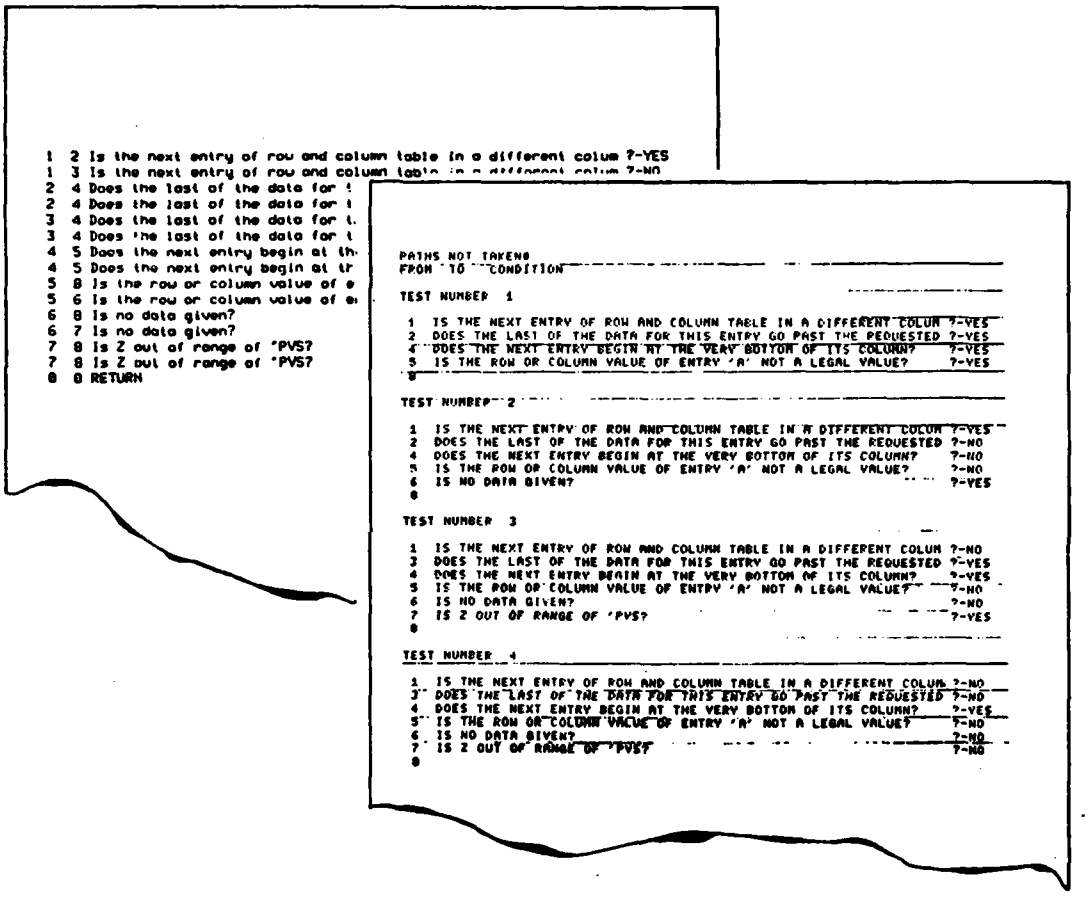
For both system developments used in this study, module testing was documented in individual programmer's notebooks. The notebooks contain the following information:

- Date of test
- Brief statement of test objectives
- Brief description of input conditions
- Description of test results and analysis

The notebooks were not available for the two system developments because of the timeframe since delivery of the two systems. A review of notebooks of current software developments revealed that sufficient information is available to determine the IN.1 measures. Several automated aids are available in this area and are discussed in paragraph 8.3. An example of automated assistance during the design phase is the minimum number of test cases to

D.17 (continued)

cover all program paths determined by GE/ISDS from design charts of a program. An example report is shown in Figure D.17-1.



```
1  2 Is the next entry of row and column table in a different colum ?-YES
1  3 Is the next entry of row and column table in a different colum ?-NO
2  4 Does the last of the data for t
2  4 Does the last of the data for t
3  4 Does the last of the data for t
3  4 Does the last of the data for t
4  5 Does the next entry begin at th
4  5 Does the next entry begin at th
5  8 Is the row or column value of e
5  6 Is the row or column value of e
6  8 Is no data given?
6  7 Is no data given?
7  8 Is Z out of range of 'PVS?
7  8 Is Z out of range of 'PVS?
8  0 RETURN
```

```
PATHS NOT TAKEN
FROM  TO   CONDITION

TEST NUMBER  1

 1  IS THE NEXT ENTRY OF ROW AND COLUMN TABLE IN A DIFFERENT COLUM ?-YES
 2  DOES THE LAST OF THE DATA FOR THIS ENTRY GO PAST THE REQUESTED ?-YES
 4  DOES THE NEXT ENTRY BEGIN AT THE VERY BOTTOM OF ITS COLUMN?    ?-YES
 5  IS THE ROW OR COLUMN VALUE OF ENTRY 'A' NOT A LEGAL VALUE?     ?-YES
 8

TEST NUMBER  2

 1  IS THE NEXT ENTRY OF ROW AND COLUMN TABLE IN A DIFFERENT COLUM ?-YES
 2  DOES THE LAST OF THE DATA FOR THIS ENTRY GO PAST THE REQUESTED ?-NO
 4  DOES THE NEXT ENTRY BEGIN AT THE VERY BOTTOM OF ITS COLUMN?    ?-NO
 5  IS THE ROW OR COLUMN VALUE OF ENTRY 'A' NOT A LEGAL VALUE?     ?-NO
 6  IS NO DATA GIVEN?                                              ?-YES
 8

TEST NUMBER  3

 1  IS THE NEXT ENTRY OF ROW AND COLUMN TABLE IN A DIFFERENT COLUM ?-NO
 3  DOES THE LAST OF THE DATA FOR THIS ENTRY GO PAST THE REQUESTED ?-YES
 4  DOES THE NEXT ENTRY BEGIN AT THE VERY BOTTOM OF ITS COLUMN?    ?-YES
 5  IS THE ROW OR COLUMN VALUE OF ENTRY 'A' NOT A LEGAL VALUE?     ?-NO
 6  IS NO DATA GIVEN?                                              ?-NO
 7  IS Z OUT OF RANGE OF 'PVS?                                     ?-YES
 8

TEST NUMBER  4

 1  IS THE NEXT ENTRY OF ROW AND COLUMN TABLE IN A DIFFERENT COLUM ?-NO
 3  DOES THE LAST OF THE DATA FOR THIS ENTRY GO PAST THE REQUESTED ?-NO
 4  DOES THE NEXT ENTRY BEGIN AT THE VERY BOTTOM OF ITS COLUMN?    ?-YES
 5  IS THE ROW OR COLUMN VALUE OF ENTRY 'A' NOT A LEGAL VALUE?     ?-NO
 6  IS NO DATA GIVEN?                                              ?-NO
 7  IS Z OUT OF RANGE OF 'PVS?                                     ?-NO
 8
```

Figure D.17-1 Minimum Test Case Generation by GE/ISDS

BEST AVAILABLE COPY

The interface control document was a primary source for identifying all of the data files used as interfaces. They are described in detail in this document. The validation and acceptance test specification indicated that testing module interfaces (IN.2(1)) is a primary objective of development testing. Included with this specification is the development test plan which covers:

- Statement of function tested
- Modules exercised
- Data base value required
- Inputs
- Expected output
- Analysis of output
- Priority of test

The performance requirements (IN.2(2)) are extracted from the software requirements specification. They are traced by a routine to insure compliance. An example output of the routine illustrates several performance requirements identified relating to a particular specification follows:

Example:

SPECIFICATION TO PERFORMANCE REQUIREMENT TRANSLATION

SPECIFICATION — The executive function shall provide the option for specific parameter status reporting and data base update.

PERFORMANCE REQUIREMENTS

EX-40

The executive will provide an operator interaction option to display the contents of specific parameters identified in the data base defined status table.

EX-41

The executive will provide an operator interaction option to data base update specific parameters identified in data base defined status table.

EX-42

The executive will provide an option to display parameter status values on the console or printer.

EX-43

The executive will accept and interpret run-control data inputs which define specific parameters to be displayed automatically, and the interval at which they are to be displayed.

EX-44

The executive will provide an option to automatically display specific parameters available for status reporting.

At the system level (IN.3), the validation and acceptance test plan provides a test requirements satisfaction matrix from which IN.3(1) can be determined.

D.18

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
SD. 1 QUANTITY OF COMMENTS (by module) $\left(\frac{\#\text{ of comments (nonblank)}}{\text{total \# lines (nonblank)}}\right)$					CODE	GJS
SD. 2 EFFECTIVENESS OF COMMENTS MEASURE						
(1) Modules have standard formated prologue comments which describe: - Module name/version number - Author - Date - Purpose - Inputs - Outputs - Function - Assumptions - Limitations and restrictions - Accuracy requirements - Error recovery procedures - References $\left(1-\frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN (CA)
(2) Comments set off from code in uniform manner. $\left(1-\frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN (CA)
(3) All transfers of control & destinations commented. $\left(1-\frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN (CA)
(4) All machine dependent code commented. $\left(1-\frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN (CA)
(5) All non-standard HOL statements commented. $\left(1-\frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN (CA)
(6) Attributes of all declared variables commented. $\left(1-\frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN (CA)
(7) Comments do not just repeat operation $\left(1-\frac{\#\text{ modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN (CA)

SD.1 was simply derived from our code audit routine, a sample output of which is shown in Figure D.18-1.

SD.2 measures were all determined manually from the source code listings but could be collected by an enhanced code audit routine. Some selected examples from the source code relating to SD.2 measurements are shown in Figure D.18-2.

TASK ID= ANNA OJSMRY SOURCE CODE 19NOV76 PAGE 15

PROGRAM NAME	MOD		CARDS	TOTAL STMNTS.	RELATIVE TO AVG.	PROCEDURAL STMNTS.	AS RATIO OF TOTAL	DECLARATIVE STMNTS.	AS RATIO OF TOTAL	COMMENTS	COMMENTS PER STMNT.	CARDS PER STMNT.
GALTR	HL	--	1419	484	1.42	404	0.83	80	0.17	332	0.69	2.93
GBECON	HB	--	2617	712	2.08	622	0.87	90	0.13	659	0.93	3.68
GBEFIT	HF	--	2582	829	2.42	698	0.84	131	0.16	563	0.68	3.11
GBESIDE	GD	--	145	25	0.07	23	0.92	2	0.08	28	1.12	5.80
GBEST	GK	--	479	114	0.33	100	0.88	14	0.12	111	0.97	4.20
GBEWARE	HG	--	1569	531	1.55	444	0.84	87	0.16	364	0.69	2.95
GBLIS	HG	--	1551	483	1.41	389	0.81	94	0.19	415	0.86	3.21
GBLOCK	HG	--	2976	1066	3.12	886	0.83	180	0.17	638	0.60	2.79
GCOM1	GB	--	242	105	0.31	92	0.88	13	0.12	62	0.59	2.30
GCOM2	GF	--	195	66	0.20	44	0.68	22	0.32	48	0.71	2.87
GCONV	GB	--	366	162	0.47	126	0.78	36	0.22	122	0.75	2.26
GCVTIME	HC	--	86	20	0.06	12	0.60	8	0.40	16	0.80	4.30
GDAHA	GB	D	341	123	0.36	84	0.68	39	0.32	113	0.92	2.77
GDBC	HE	D	919	350	1.02	319	0.91	31	0.09	242	0.69	2.63
GDBCC	HC	D	439	162	0.47	144	0.89	18	0.11	95	0.59	2.71
GDBCCF	HC	D	1341	472	1.38	425	0.90	47	0.10	333	0.71	2.84
GDBCM	HC	D	383	149	0.44	135	0.91	14	0.09	117	0.79	2.57
GDBCT	HC	D	1379	531	1.55	485	0.91	46	0.09	428	0.81	2.60
GDBD	HE	D	607	222	0.65	197	0.89	25	0.11	85	0.38	2.73
GDBDS	HF	D	1242	493	1.44	428	0.87	65	0.13	400	0.81	2.52
GDBDY	HC	D	2231	999	2.92	880	0.88	119	0.12	927	0.93	2.23
GDBE	HB	D	327	173	0.51	151	0.87	22	0.13	54	0.31	1.89
GDBED	HB	D	2169	953	2.79	840	0.88	113	0.12	532	0.56	2.28
GDBER	HB	D	1165	589	1.72	538	0.91	51	0.09	124	0.21	1.98

Figure DJ8-1. Comment Count By Code Audit Routine

BEST AVAILABLE COPY

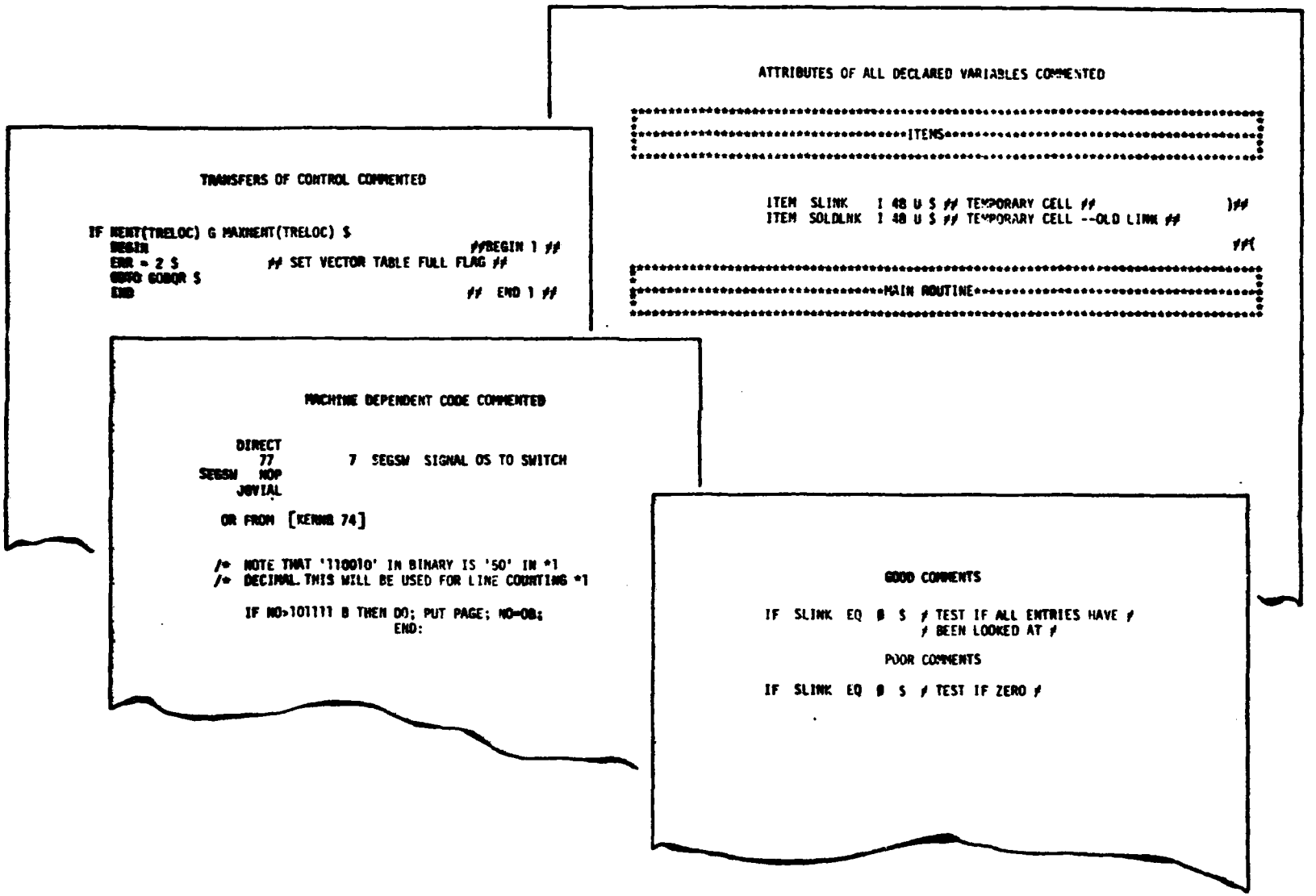


Figure D.18-2 Effectiveness of Comments Examples

D.19

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
SD. 3 DESCRIPTIVENESS OF IMPLEMENTATION LANGUAGE MEASURE						
(1) High order language used. $\left(1 - \frac{\text{\# modules with direct code}}{\text{total \# modules}}\right)$					CODE	GJS
(2) Standard format for organization of modules. $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN (CA)
(3) Variable names (mnemonic) descriptive of physical or functional property represented. $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN
(4) Source code logically blocked and indented. $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN
(5) One statement per line. $\left(1 - \frac{\text{\# continuations + multiple statement lines}}{\text{total \# lines}}\right)$					CODE	GJS
(6) No language keywords used as names. $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN (CA)

SD.3(1) and (5) were collected by the code audit routine. The remainder were determined manually. An example of compliance with SD.3(3) follows:

```
ITEM  SAVBLK   H   8  $  ≠≠  SAVE BLOCK NAME  ≠≠
ITEM  SAVFILE  I  48  U  $  ≠≠  SAVE FILE NUMBER  ≠≠
ITEM  SAVREC   I  48  U  $  ≠≠  SAVE RECORD NUMBER  ≠≠
ITEM  OPERANS  H   8  $  ≠≠  OPERATOR RESPONSE  ≠≠
```

which are obviously better than variable names such as AØØ1, AØØ2, AØØ3, etc.

But more subtle problems can arise from poor variable names as pointed out in this example in [KERNB74].

```
      N = K
      N = K**2
    NNN = K**3
     .
     .
     .
  WRITE (6,60) N,NN,NNN
```

B.19 (continued)

In this example the typographical error in the second line would be easy to make and difficult to find because of the naming. A suggested naming scheme is N, NSQ, NCUBE.

Most structured programming preprocessors provide the indentation emphasized in SD.3(4). Also, the paragraphing constructs such as DO ... CONTINUE in FORTRAN, BEGIN ... END in ALGOL and JOVIAL, the DO; ... END;, BEGIN; ... END; and PROCEDURE; ... END; in PL/1 should be taken advantage of to promote self-descriptiveness. Whether a module is implemented with these techniques in mind or not, is easily discernible by inspection of the source code.

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
EE. 1 PERFORMANCE REQUIREMENTS ALLOCATED TO DESIGN			PDS DDS	RTR (PSL) (PDL)		
EE. 2 ITERATIVE PROCESSING EFFICIENCY MEASURE: (by module)						
(1) Non-loop dependent computations kept out of loop. $\left(1 - \frac{\#\text{ nonloop dependent statements in loop}}{\text{total \# loop statements}}\right)$			DDS	MAN	CODE	MAN (CA)
(2) Performance optimizing compiler/assembly language used.					CODE	MAN GJS
(3) Compound expressions defined once. $\left(1 - \frac{\#\text{ compound expression defined more than once}}{\#\text{ compound expressions}}\right)$					CODE	MAN (CA)
(4) Number of overlays. $\left(\frac{1}{\#\text{ of overlays}}\right)$			DDS	MAN	CODE	MAN (EA)
(5) Free of bit/byte packing/unpacking in loops.			DDS	MAN	CODE	MAN (CA)
(6) Free of nonfunctional executable code. $\left(1 - \frac{\#\text{ nonfunctional executable code}}{\text{total executable statements}}\right)$					CODE	MAN
(7) Decision statements efficiently coded. $\left(1 - \frac{\#\text{ inefficient decision statements}}{\text{total \# decision statements}}\right)$					CODE	MAN (EA)
(8) Module linkages. $\left(1 - \frac{\text{module linkage time}}{\text{execution time}}\right)$					CODE	(EA)
(9) OS linkages. $\left(1 - \frac{\text{OS linkage time}}{\text{execution time}}\right)$					CODE	(EA)
EE. 3 DATA USAGE EFFICIENCY MEASURE: (by module)						
(1) Data grouped for efficient processing.			DDS DBMP	MAN	CODE	MAN (DBO)
(2) Variables initialized when declared. $\left(\frac{\#\text{ initialized when declared}}{\text{total \# variables}}\right)$					CODE	MAN (CA)
(3) No mix-mode expressions. $\left(1 - \frac{\#\text{ mix mode expressions}}{\#\text{ executable statements}}\right)$					CODE	MAN (CA)
(4) Common choice of units/type. $(1/\#\text{ occurrences of uncommon unit operations})$			DDS	MAN	CODE	MAN (CA)
(5) Data indexed or referenced for efficient processing.			DDS DBMP	MAN	CODE	MAN (DBO) (CA)

D.20 (continued)

Paragraph D.17 illustrated how the performance requirements are kept track of and paragraph D-1 described the SRS requirements satisfaction vs routine matrix in the preliminary design specification. These sources identify the performance requirements, and the detailed design specifications reveal whether they are accommodated (EE.1). For example, estimates of storage requirements and run times are documented in the design documents. These estimates which provide a best estimate, as well as a high, and a low versus the maximum allowable are based on specific scenarios. The SRS might specify a certain transaction type must be handled within a certain time frame. The run time estimates for the series of modules which process that transaction type should comply with the requirement.

Six of the measures for EE.2 and EE.3 can be taken from the detailed design documents. All of the measures can be determined from the source code. Code audit, execution analysis, and data base analysis all contribute to measurements for these metrics. The most common violations to the efficiency-oriented characteristics that these metrics represent were related to EE.2(1) and (3). Some examples are:

```
FOR I  =  1,N
   BEGIN
   AREA(I) = 2*PI*(I+X)**2
   END
```

The 2*PI is an unnecessary computation within a loop.

As illustrated in [VANTD74] the following:

```
SIGMA1 = SIN(THETA) + SIN(THETA)**2
SIGMA2 = SIN(THETA)/3.0
```

is much more efficient if implemented as:

```
RHO = SIN(THETA)
SIGMA1 = RHO+RHO**2
SIGMA2 = RHO/3.0
```

D.21

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
SE. 1 STORAGE EFFICIENCY MEASURE: (by module)						
(1) Storage requirements allocated to design.			DDS	MAN (PSL) (PDL)		
(2) Virtual storage facilities used.			DDS	MAN	CODE	MAN
(3) Common data defined only once. $\left(1 - \frac{\text{\# variables defined more than once}}{\text{total \# variables}}\right)$					CODE	MAN
(4) Program segmentation. $\left(1 - \frac{\text{maximum segment length}}{\text{total program length}}\right)$			DDS	MAN	CODE	(EA)
(5) Data segmentation. $\left(1 - \frac{\text{amount of unused data}}{\text{total amount of data}}\right)$			DDS	MAN	CODE	(EA)
(6) Dynamic memory management utilized.			DDS	MAN	CODE	MAN
(7) Data packing used.					CODE	MAN (CA)
(8) Free of nonfunctional code. $\left(1 - \frac{\text{\# nonfunctional statements}}{\text{total \# statements}}\right)$					CODE	MAN
(9) No duplicate codes. $\left(1 - \frac{\text{\# duplicate statements}}{\text{total \# statements}}\right)$			CODE	MAN	CODE	MAN
(10) Storage optimizing compiler/assembly language used.					CODE	MAN GJS
(11) Free of redundant data elements. $\left(1 - \frac{\text{\# redundant data elements}}{\text{\# data elements}}\right)$					CODE	MAN

The programming environment for Systems A and B did not provide capabilities such as virtual storage, optimizing compilers, or dynamic memory management. However, a very restrictive amount of memory available encouraged many efficiency techniques to be utilized. These measures attempt to identify poor practices.

METRIC		REQMTS		DESIGN		IMPLEMENTATION	
		SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
AC. 1	ACCESS CONTROL CHECKLIST:						
	(1) User I/O access controls provided (ID's, passwords).	SRS	MAN	PDS DDS	MAN	CODE	MAN
	(2) Data base access control provided (authorization tables, privacy locks).	SRS	MAN	PDS DDS	MAN	CODE	MAN
	(3) Memory protection across tasks provided.	SRS	MAN	PDS DDS	MAN	CODE	MAN
AA. 1	ACCESS AUDIT CHECKLIST:						
	(1) Provisions for recording and reporting access.	SRS	MAN	PDS DDS	MAN	CODE	MAN
	(2) Provisions for immediate indication of access violation.	SRS	MAN	PDS DDS	MAN	CODE	MAN

Neither System A nor B had any specified requirement for access control capabilities. Therefore, each of these measures were not applicable. The identified sources would normally contain statements which would provide the required data.

D.23

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
OP. 1 OPERABILITY CHECKLIST:						
(1) All steps of operation described (normal and alternative flows).	SRS	MAN	UOM	MAN	UOM	MAN
(2) All error conditions and responses appropriately described to operator.	SRS	MAN	UOM	MAN	UOM	MAN
(3) Provisions for operator to interrupt, obtain status, save, modify, and continue processing.	SRS	MAN	UOM	MAN	UOM	MAN
(4) Number of operator actions reasonable. $\left(1 - \frac{\text{time for operator actions}}{\text{total time for job}}\right)$					CODE	(EA)
(5) Job set up and tear down procedures described.					UOM	MAN
(6) Hard copy log of interactions maintained.			UOM	MAN	UOM	MAN
(7) Operator messages consistent and responses standard.			UOM	MAN	UOM	MAN

The user's manual or operator manual (which we call the Computer Programs Operating Instructions) is the primary source for these measures. Generally the user's manual evolves during the development phase, becoming more detailed and precise as more detailed information on the exact operating procedures becomes available. A separate section contains the daily usage instructions and another section identifies and explains each error message.

A specific example, paraphrased from the SRS of System B is the following itemized requirement:

> The system should recognize error conditions and automatically abort in an unrecoverable situation, or in a situation where recovery is possible, allow operator intervention to either:
>
> - abort the program
> - accept error and continue the program
> - correct error and continue the program.

D.24

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
TR. 1 TRAINING CHECKLIST:						
(1) Lesson plans/training material developed for operators, end users, maintainers.					TM	MAN
(2) Realistic simulated exercises provided.					TM	MAN
(3) Sufficient 'help' and diagnostic information available on-line.			DDS	MAN	UOM	MAN (EA)

Normally a training manual, course material, and lesson plans would be available to evaluate and provide a measures for this metric. In the environment of the two systems of this study, personnel quite familiar with the systems operated them. Thus much less formal documentation was available. The user's manual had a section on training and introductory information on the systems. These measures are manually extracted from the identified sources.

An excellent example of compliance with the TR.1(3) measure is found in the support software, *GE/ISDS*. At any level of the system, *'HELP'* can be typed, and a list of correct commands at that level is provided the on-line user.

D.25

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
CM. 1 USER INPUT INTERFACE MEASURE:						
(1) Default values defined. $\left(\frac{\#\ \text{defaults}}{\text{total}\ \#\ \text{parameters}}\right)$			DDS	MAN	UOM	MAN
(2) Input formats uniform. $\left(\frac{1}{\#\ \text{different input record formats}}\right)$			DDS	MAN	UOM	MAN
(3) Each input record self identifying. $\left(1-\frac{\#\ \text{that are not self identifying}}{\text{total}\ \#\ \text{input records}}\right)$			DDS	MAN	UOM	MAN
(4) Input can be verified by user prior to execution.			DDS	MAN	UOM	MAN
(5) Input terminated by explicitly defined logical end of input.			DDS	MAN	UOM	MAN
(6) Provision for specifying input from different media.	SRS	MAN	DDS	MAN	UOM	MAN
CM. 2 USER OUTPUT INTERFACE MEASURE:						
(1) Selective output controls.	SRS	MAN	DDS	MAN	UOM	MAN
(2) Outputs have unique descriptive user oriented labels.			DDS	MAN	UOM	MAN
(3) Outputs have user oriented units.			DDS	MAN	UOM	MAN
(4) Uniform Output formats. $\left(\frac{1}{\#\ \text{different output formats}}\right)$			DDS	MAN	UOM	MAN
(5) Logical groups of output separated for user examination.			DDS	MAN	UOM	MAN
(6) Relationship between error messages and outputs is unambiguous.			DDS	MAN	UOM	MAN
(7) Provision for reducing output to different media.			DDS	MAN	UOM	MAN

Manual reviews of the detailed design specifications and users manual provided the data for all of these measures. Each input/output format was described in the users manual. The SRS identified the requirements for different modes of operation (which involved different I/O), from cards, tape, and teletype, and for output compression and expansion.

D.25 (continued)

The selective output controls should include selective debug options in case the program exhibits unusual behavior. The output labels and units are almost as valuable to the user as the results themselves. Error messages should be helpful, for example:

```
PARAMETER ******* MUST BE BCI
LOGICAL UNIT *** IS NOT VALID
DATA BASE **** WAS NOT FOUND IN THE DISC DIRECTORY
VERB ****** IS NOT LEGAL FOR FUNCTION.
```

D.26

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
SS. 1 SOFTWARE SYSTEM INDEPENDENCE MEASURE:						
(1) Dependence on software system utility programs. $\left(1 - \frac{\#\text{ programs = utility program}}{\text{total \# programs}}\right)$			DDS	MAN	CODE	MAN (CA)
(2) Dependence on software system library routines. $\left(1 - \frac{\#\text{ library routines used}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN (CA)
(3) Common, standard subset of language used. $\left(1 - \frac{\#\text{ module violate rule}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN (CA)
(4) Free from operating system references. $\left(1 - \frac{\#\text{ modules with OS references}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN (CA)

These measures were taken manually from the code and design documents. Evaluation of the execution or compilation of a routine also is helpful. In a particular environment, automated identification of these measures would be possible.

D.27

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
MI. 1 MACHINE INDEPENDENCE MEASURE:						
(1) Programming language used available on other machines.			DDS	MAN	CODE	MAN
(2) Free from input/output references. $\left(1 - \frac{\text{\# modules with I/O references}}{\text{total \# modules}}\right)$			DDS	MAN	CODE	MAN (CA)
(3) Code is independent of word and character size $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN
(4) Data representation machine independent. $\left(1 - \frac{\text{\# modules violate rule}}{\text{total \# modules}}\right)$					CODE	MAN

Specific JOVIAL constructs such as BIT/BYTE, the I/O system routines, and DIRECT code were keyed on while searching the code to determine these measures.

D.28

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
CC. 1 COMMUNICATIONS COMMONALITY CHECKLIST:						
(1) Definitive statement of requirement for communication with other systems.	SRS	MAN				
(2) Protocol standards established and followed.			ICD DDS SC	MAN	CODE	MAN
(3) Single module interface for input. $\left(\frac{1}{\#\text{ modules used for input}}\right)$			DDS	MAN (PDL)	CODE	MAN (CA)
(4) Single module interface for output. $\left(\frac{1}{\#\text{ modules used for output}}\right)$			DDS	MAN (PDL)	CODE	MAN (CA)

The interface control document specifically identifies and describes any interfaces between systems. This is especially important in an associate contractor environment. The detail design specification and the code were utilized to identify the routines which contained any I/O operations.

D.29

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
DC. 1 DATA COMMONALITY CHECKLIST:						
(1) Definitive statement for standard data representation for communication with other systems.	SRS	MAN				
(2) Translation standards among representations established and followed.			ICD DDS SC	MAN	CODE	MAN (CA)
(3) Single module to perform each translation. $\left(\frac{1}{\#\text{ modules used to perform translation}}\right)$			DDS	MAN	CODE	MAN

There was no requirements for communication between systems as envisioned by this metric and therefore data was not collected.

D.30

METRIC	REQMTS		DESIGN		IMPLEMENTATION	
	SOURCE	TOOL	SOURCE	TOOL	SOURCE	TOOL
CO. 1 HALSTEAD'S MEASURE (by module) $\left(1-\left\lvert\frac{\text{module length calculated-module length observed}}{\text{module length observed}}\right\rvert\right)$					CODE	GJS

The number of operators and operands were collected by the code audit routine. A routine was written to calculate the metric based on those inputs.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

Printed by
United States Air Force
Hanscom AFB, Mass. 01731